ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση Υπερφασματικής Απεικόνισης με χρήση του εργαλείου Apache Spark σε Κατανεμημένο Περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διονυσία Α. Βούλγαρη

Κωνσταντίνος Χ. Τζιώτης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβριος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση Υπερφασματικής Απεικόνισης με χρήση του εργαλείου Apache Spark σε Κατανεμημένο Περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διονυσία Α. Βούλγαρη

Κωνσταντίνος Χ. Τζιώτης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31η Οκτωβρίου 2018.

.....	.....	.....
Θ. Βαρβαρίγου	Σ. Παπαβασιλείου	Δ. Ασκούνης
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2018

.....

Διονυσία Α. Βούλγαρη
Κωνσταντίνος Χ. Τζιώτης

Διπλωματούχοι Ηλεκτρολόγοι Μηχανικοί και Μηχανικοί Υπολογιστών Ε.Μ.Π.

Περιληψη

Τα τελευταία χρόνια υπάρχει έντονη αύξηση των υπερφασματικών δεδομένων που συλλέγονται από δορυφόρους προς επεξεργασία και αποθήκευση. Τα δεδομένα αυτά χαρακτηρίζονται από υψηλό όγκο, καθώς εμπεριέχουν μεγάλη και αναλυτική πληροφορία για την περιοχή παρατήρησης σε ολόκληρο το ηλεκτρομαγνητικό φάσμα. Η ανάλυση των πληροφοριών αυτών είναι χρήσιμη για την εξαγωγή συμπερασμάτων σχετικά με τη μορφολογία του εδάφους και τη σύστασή του. Για το σκοπό αυτό απαιτείται η χρήση εργαλείων και τεχνολογιών που μπορούν να διαχειριστούν αποδοτικά μεγάλα δεδομένα(Big Data).

Η παρούσα διπλωματική εργασία καλείται να επιλύσει αποδοτικά το πρόβλημα του φασματικού διαχωρισμού(Spectral Unmixing), το οποίο αποτελεί ένα σύνθετο πρόβλημα ανάλυσης υπερφασματικών δεδομένων με στόχο την αναγνώριση των υλικών και της ποσόστωσής τους σε κάθε εικονοστοιχείο(pixel) της εικόνας. Για την υλοποίηση του μοντέλου επίλυσης, χρησιμοποιήθηκε το εργαλείο Apache Spark και το κατανεμημένο σύστημα αρχείων HDFS(Hadoop Distributed File System). Η εφαρμογή που κατασκευάστηκε χωρίζεται σε τρία βασικά μέρη, τα οποία αναλαμβάνουν τον προσδιορισμό του πλήθους των καθαρών υλικών(endmembers), την εξαγωγή των φασματικών υπογραφών τους και την εύρεση της ποσόστωσής τους σε κάθε pixel. Πραγματοποιήθηκαν πειράματα για την επαλήθευση των αποτελεσμάτων κάθε σταδίου με χρήση συνθετικών δεδομένων, τα οποία κατασκευάστηκαν από υλικά της φασματικής βιβλιοθήκης του USGS. Τέλος, δόθηκε ιδιαίτερη έμφαση στην ανάλυση της επίδοσης της εφαρμογής, μέσω της παραμετροποίησης των πόρων που χρησιμοποιεί το Apache Spark σε ένα κατανεμημένο σύστημα.

Λέξεις κλειδιά

Υπερφασματικά δεδομένα, Spectral Unmixing, Endmember Extraction, Abundance Estimation, Apache Spark, Big Data, HDFS, Geotrellis.

ABSTRACT

In recent years, there has been a significant increase in the amount of hyperspectral data collected by satellites for processing and storage. These data are characterized by a high volume as they contain detailed information about the observation area across the entire electromagnetic spectrum. The analysis of this information is useful for drawing conclusions on soil morphology and composition. This requires the use of tools and technologies that can efficiently handle large data (Big Data).

This diploma thesis is intended to solve the Spectral Unmixing problem, which is a complex problem of the analysis of hyperspectral data in order to identify the materials and their quantification in each pixel of the image. To implement the resolving model, the Apache Spark tool and the Hadoop Distributed File System (HDFS) were used. The built-in application is divided into three main parts, which are responsible for determining the number of endmembers, extracting their spectral signatures, and finding their quota in each pixel. Experiments were performed to verify the results of each step using synthetic data generated with materials from the USGS spectral library. Finally, particular emphasis was placed on analyzing the performance of the application through the configuration of resources used by Apache Spark in a distributed system.

Key words

Hyperspectral data, Spectral Unmixing, Endmember Extraction, Abundance Estimation, Apache Spark, Big Data, HDFS, Geotrellis.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Κατανεμημένης γνώσης και Συστημάτων Πολυμέσων του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Στο σημείο αυτό θα θέλαμε να απευθύνουμε θερμές ευχαριστίες στην επιβλέπουσα καθηγήτρια κ. Θεοδώρα Βαρβαρίγου για την ευκαιρία που μας έδωσε να ασχοληθούμε με ένα σύγχρονο θέμα υψηλού ενδιαφέροντος. Στον υποψήφιο Διδάκτορα του Ε.Μ.Π. Βρεττό Μουλό, στο Γιώργο Χατζηκυριάκο και στο Δημήτρη Συκά χρωστάμε ένα μεγάλο ευχαριστώ για την επιστημονική καθοδήγησή, τις συμβουλές και τη βοήθεια που μας προσέφεραν κατά τη διάρκεια πραγματοποίησης αυτής της διπλωματικής εργασίας.

Ιδιαίτερες ευχαριστίες θα θέλαμε να απευθύνουμε στη Γεωργία, στην Ειρήνη, στην Ελευθερία, στη Μαλβίνα, στο Σπύρο και στο Φαίδωνα για την υπομονή και συμπαράστασή τους, καθώς χωρίς αυτούς η εκπόνηση της παρούσας εργασίας δε θα ήταν τόσο ομαλή. Τέλος, θα θέλαμε να ευχαριστήσουμε τους συγγενείς και κοντινούς μας φίλους για την υποστήριξη και την εμπιστοσύνη που μας έδειξαν καθόλη τη διάρκεια των σπουδών μας.

Διονυσία Α. Βούλγαρη και Κωνσταντίνος Χ. Τζιώτης,
Αθήνα, Οκτώβριος 2018

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1 – ΕΙΣΑΓΩΓΗ

1.1. Περιγραφή του προβλήματος

Στην σημερινή εποχή ο όγκος δεδομένων πληροφορίας έχει αυξηθεί αισθητά. Αντίστοιχα έχουν πολλαπλασιαστεί και τα προβλήματα που καλούνται να αναλύσουν και να επεξεργαστούν αυτά τα δεδομένα. Τα τελευταία χρόνια υπάρχει μεγάλη ανάπτυξη εφαρμογών που διαχειρίζονται δορυφορικά δεδομένα, τα οποία συλλέγονται καθημερινά και μάλιστα με πολύ υψηλό ρυθμό. Για την επεξεργασία αυτών των δεδομένων είναι αναγκαία τόσο εξελιγμένα συστήματα αποθήκευσης όσο και εργαλεία επεξεργασίας μεγάλων δεδομένων(Big Data). Από το σύνολο των δεδομένων που συλλέγονται από δορυφόρους, η παρούσα διπλωματική εργασία καλείται να διαχειριστεί υπερφασματικές απεικονίσεις, οι οποίες περιέχουν πολύ μεγάλο όγκο πληροφορίας, καθώς περιλαμβάνουν αναλυτικές λεπτομέρειες σε ολόκληρο το ηλεκτρομαγνητικό φάσμα.

Οι υπερφασματικές απεικονίσεις αναλύονται μέσω μίας πληθώρας αλγορίθμων, οι οποίοι έχουν αυξημένη πολυπλοκότητα. Η ανάλυση αυτή αποτελείται από πολλά στάδια, τα οποία εμπεριέχουν υψηλό υπολογιστικό φορτίο εργασίας. Αρχικά, είναι αναγκαία η προεπεξεργασία των δεδομένων που συλλέγονται από τους δορυφόρους, με στόχο την εξάλειψη σφαλμάτων εξαιτίας της ατμόσφαιρας και τυχόν λαθών κατά τη διαδικασία σάρωσης από τον αισθητήρα. Στην συνέχεια εφαρμόζονται στα φιλτραρισμένα δεδομένα διάφοροι αλγόριθμοι, οι οποίοι αναλαμβάνουν τόσο την αναγνώριση της μορφολογίας του εδάφους όσο και την ταυτοποίηση συγκεκριμένων υλικών σε αυτά. Ένα από τα σημαντικότερα προβλήματα στην ανάλυση υπερφασματικών απεικονίσεων είναι η διαδικασία του Spectral Unmixing, μέσω της οποίας εξάγονται πληροφορίες για την περιεκτικότητα των υλικών σε κάθε εικονοστοιχείο(pixel) μίας εικόνας. Η διαδικασία του Spectral Unmixing εφαρμόζεται σε δεδομένα πολύ μεγάλου όγκου και απαιτεί τεχνολογίες ικανές να εκτελέσουν τους περίπλοκους αλγορίθμους από τους οποίους απαρτίζεται.

Στην παρούσα εργασία θα χρησιμοποιηθεί το εργαλείο Apache Spark για τον σχεδιασμό και την υλοποίηση μίας εφαρμογής που θα αναλάβει να επιλύσει το πρόβλημα του Spectral Unmixing σε ένα κατανεμημένο σύστημα. Το Apache Spark είναι ένα εργαλείο διαχείρισης Big Data με κατανεμημένο τρόπο και αποτελεί την καταλληλότερη επιλογή για την αποδοτική αντιμετώπιση του συγκεκριμένου προβλήματος.

1.2. Οργάνωση του Κειμένου

Το Κεφάλαιο 2 παρουσιάζει τον ορισμό της υπερφασματικής απεικόνισης και τους τομείς στους οποίους χρησιμοποιείται. Δίνονται αναλυτικά οι τρόποι με τους οποίους συλλέγεται ένας υπερφασματικός κύβος, καθώς και η προεπεξεργασία που απαιτείται να εφαρμοστεί σε αυτόν. Στην συνέχεια, αναλύεται το πρόβλημα του φασματικού διαχωρισμού και εξηγούνται τα στάδια από τα οποία αποτελείται.

Το Κεφάλαιο 3 ασχολείται με το εργαλείο Apache Spark, δίνοντας πληροφορίες σχετικά με την αρχιτεκτονική του, το πυρήνα και τις βιβλιοθήκες του. Ακόμη, εξηγείται η διαδικασία εκτέλεσης μίας εφαρμογής στο Spark, καθώς και οι τρόποι παραμετροποίησης και παρακολούθησης αυτής από το χρήστη.

Το Κεφάλαιο 4 αναλύει την πλατφόρμα υλοποίησης της εφαρμογής που σχεδιάστηκε και δημιουργήθηκε στην παρούσα εργασία. Αναλύονται, εκτενώς, οι αλγόριθμοι που χρησιμοποιήθηκαν, η υλοποίησή τους, καθώς και διάφορα πειράματα τα οποία επαληθεύουν την ορθότητα της εφαρμογής.

Στο Κεφάλαιο 5 παρουσιάζονται πειράματα και μετρήσεις, που αποσκοπούν στην ανάλυση της επίδοσης της εφαρμογής. Εξηγείται διεξοδικά ο τρόπος με τον οποίο επηρεάζεται ο χρόνος εκτέλεσης της εφαρμογής από διάφορους παράγοντες του συστήματος.

Τέλος, το Κεφάλαιο 6 περιλαμβάνει γενικά συμπεράσματα, καθώς και μελλοντικές επεκτάσεις της παρούσας εργασίας.

ΚΕΦΑΛΑΙΟ 2 – ΥΠΕΡΦΑΣΜΑΤΙΚΗ ΑΠΕΙΚΟΝΙΣΗ

2.1. Υπερφασματική Απεικόνιση και Εφαρμογές της

Τα υπερφασματικά δεδομένα που συλλέγονται σήμερα από αισθητήρες αυξάνονται ραγδαία σε όγκο καθώς επιτρέπουν την απόκτηση αναλυτικής φυσικής πληροφορίας της περιοχής παρατήρησης. Η υπερφασματική απεικόνιση, όπως και άλλες φασματικές απεικονίσεις, συλλέγει και επεξεργάζεται πληροφορίες από ολόκληρο το ηλεκτρομαγνητικό φάσμα. Στόχος της είναι η απόκτηση του φάσματος για κάθε στοιχείο στην εικόνα μίας περιοχής, με σκοπό την εύρεση και εντοπισμό αντικειμένων, την ταυτοποίηση υλικών καθώς και την ανίχνευση περιβαλλοντικών μεταβολών. Ενώ το ανθρώπινο μάτι βλέπει το χρώμα του ορατού φωτός σε τρεις κυρίως ζώνες (μεγάλα μήκη κύματος - αντιληπτά ως κόκκινα, μεσαία μήκη κύματος – αντιληπτά ως πράσινα και μικρά μήκη κύματος - αντιληπτά ως μπλε), η φασματική απεικόνιση διαιρεί το φάσμα σε πολλές άλλες μικρότερες ζώνες.

Αυτή η τεχνική διαίρεσης εικόνων σε ζώνες μπορεί να χρησιμεύσει σε μία ευρεία σειρά εφαρμογών. Η δυνατότητα εντοπισμού διαφόρων υλικών που προσφέρει η υπερφασματική απεικόνιση την καθιστά ιδανική για τα ορυχεία και τις βιομηχανίες πετρελαίου στην εύρεση μεταλλευμάτων και πετρελαίου. Αν και αναπτύχθηκε αρχικά για την εξόρυξη ορυκτών και τη γεωλογία, πλέον έχει εξαπλωθεί σε πεδία όπως η γεωργία, η υγιεινή και φροντίδα των ματιών, η επεξεργασία τροφίμων, η ορυκτολογία και η προστασία του περιβάλλοντος.

Γεωργία

Αν και το κόστος της απόκτησης υπερφασματικών εικόνων είναι συνήθως υψηλό, η χρήση υπερφασματικής τηλεπισκόπησης αυξάνεται για την παρακολούθηση της ανάπτυξης και της υγείας συγκεκριμένων καλλιεργειών που βρίσκονται σε συγκεκριμένα κλίματα. Στην Αυστραλία για παράδειγμα, είναι σε εξέλιξη εργασίες για τη χρήση υπερφασματικών αισθητήρων απεικόνισης με στόχο την ανίχνευση της ποικιλίας σταφυλιών και την έγκαιρη προειδοποίηση για την εμφάνιση ασθενειών. Επιπλέον, γίνεται επεξεργασία υπερφασματκών δεδομένων για την ανίχνευση της χημικής σύνθεσης των φυτών, με την οποία παρακολουθείται σε τι κατάσταση θρεπτικών συστατικών και ενυδάτωσης βρίσκονται οι καλλιέργειες σιταριού. Σε μικρότερη κλίμακα, η υπερφασματική απεικόνιση μπορεί να χρησιμοποιηθεί για την παρακολούθηση της εφαρμογής φυτοφαρμάκων σε μεμονωμένους σπόρους και για τον ποιοτικό έλεγχο της βέλτιστης δόσης και της ομοιογενούς κάλυψης.

Μια άλλη εφαρμογή στη γεωργία είναι η ανίχνευση ζωικών πρωτεϊνών σε σύνθετες ζωοτροφές για την αποφυγή της σπογγώδους εγκεφαλοπάθειας των βοοειδών, γνωστή και ως ασθένεια των τρελών αγελάδων. Συγκεκριμένα, παρασκευάστηκαν υπερφασματικές βιβλιοθήκες που είναι αντιπροσωπευτικές της ποικιλίας των συστατικών που συνήθως υπάρχουν στην παρασκευή σύνθετων ζωοτροφών.

Υγιεινή και Φροντίδα των Ματιών

Ερευνητές του Πανεπιστημίου του Μόντρεαλ συνεργάζονται με την Photon και την Optina Diagnostics για να ελέγξουν κατά πόσο μπορεί η υπερφασματική απεικόνιση να βοηθήσει στη διάγνωση της αμφιβληστροειδοπάθειας και του οιδήματος της ωχράς κηλίδας πριν εμφανιστεί βλάβη στο μάτι. Ο υπερευαίσθητος υπερφασματικός αισθητήρας μπορεί να ανιχνεύσει μια πτώση στην κατανάλωση οξυγόνου στον αμφιβληστροειδή, γεγονός που υποδηλώνει πιθανή ασθένεια.

Επεξεργασία Τροφίμων

Στη βιομηχανία επεξεργασίας τροφίμων, η υπερφασματική απεικόνιση επιτρέπει τον εντοπισμό και την αφαίρεση ελαττωματικών και ξένων υλικών που είναι αόρατα για τους παραδοσιακούς διαλογείς φωτογραφικών μηχανών και λέιζερ. Η χρήση της υπερφασματικής απεικόνισης σε ψηφιακούς διαλογείς πετυχαίνει την επιθεώρηση μεγάλου όγκου παραγόμενων προϊόντων σε μικρό χρόνο. Ο χρήστης καθορίζει τα όρια αποδοχής και απόρριψης υλικών και το σύστημα αφαιρεί αυτόματα τα μη αποδεκτά υλικά. Η πρόσφατη εμπορική υιοθέτηση υπερφασματικών συσκευών διαλογής τροφίμων, βασισμένων σε αισθητήρες, είναι ιδιαίτερα εξελιγμένη στη βιομηχανία ξηρών καρπών όπου τα εγκατεστημένα συστήματα βελτιώνουν την απομάκρυνση των λίθων, των κελυφών και άλλων ξένων φυτικών ουσιών από καρύδια, αμύγδαλα, φιστίκια και άλλους ξηρούς καρπούς. Τέλος, η είσοδος υπερφασματικών ταξινομητών στη βιομηχανία της πατάτας επιτυγχάνει την εύκολη ανίχνευση προβλημάτων, τα οποία οι κλασικοί επεξεργαστές δυσκολεύονται να αντιληφθούν.

Ορυκτολογία

Τα γεωλογικά δείγματα μπορούν να χαρτογραφηθούν εύκολα με τη χρήση υπερφασματικής απεικόνισης. Η υπερφασματική τηλεπισκόπηση είναι ιδιαίτερα αναπτυγμένη στον εντοπισμό ορυκτών από δορυφορικές φωτογραφίες, με αποτέλεσμα τον εντοπισμό πολύτιμων λίθων, όπως ο χρυσός και τα διαμάντια.

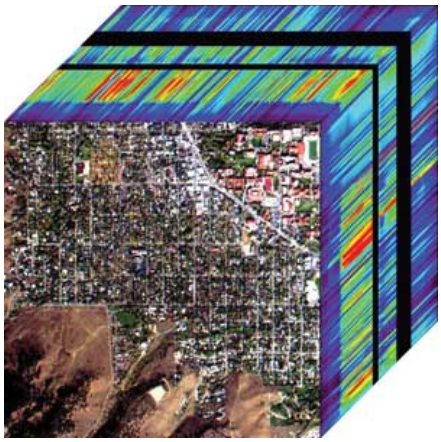
Προστασία του Περιβάλλοντος

Στις περισσότερες χώρες απαιτείται η συνεχή παρακολούθηση των εκπομπών που παράγονται από σταθμούς παραγωγής ηλεκτρικής ενέργειας(άνθρακα και πετρελαίου), από εργοστάσια τσιμέντου, καθώς και από πολλούς άλλους τύπους βιομηχανικών πηγών. Αυτή η παρακολούθηση συνήθως εκτελείται χρησιμοποιώντας συστήματα εκχυλιστικής δειγματοληψίας σε συνδυασμό με τεχνικές υπέρυθρης φασματοσκοπίας. Ορισμένες πρόσφατες απομακρυσμένες μετρήσεις που πραγματοποιήθηκαν επέτρεψαν την αξιολόγηση της ποιότητας του αέρα.

Επομένως, είναι φανερό ότι οι μηχανικοί κατασκευάζουν υπερφασματικούς αισθητήρες και συστήματα επεξεργασίας για εφαρμογές που σχετίζονται τόσο με επιστημονικές, όσο και με αναπτυξιακές δραστηριότητες.

2.2. Τεχνικές Σάρωσης Υπερφασματικού Κύβου

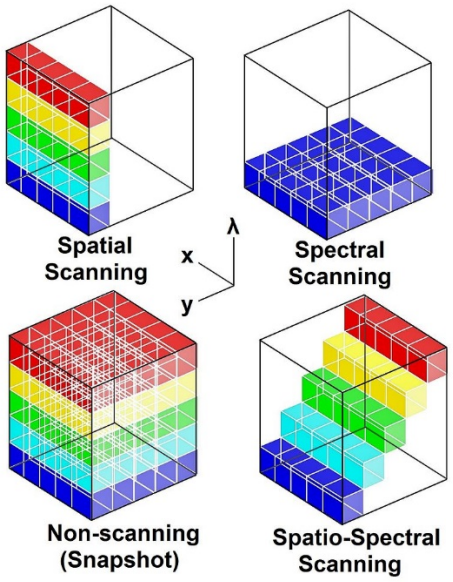
Οι υπερφασματικοί αισθητήρες συλλέγουν πληροφορία ως ένα σύνολο εικόνων. Κάθε εικόνα αντιπροσωπεύει ένα στενό εύρος μήκους κύματος του ηλεκτρομαγνητικού φάσματος, γνωστό ως φασματική ζώνη. Αυτές οι εικόνες συνδυάζονται για να σχηματίσουν έναν τρισδιάστατο κύβο υπερφασματικών δεδομένων (Εικόνα 1) με διαστάσεις (x, y, L), όπου τα x και y αντιπροσωπεύουν δύο χωρικές διαστάσεις της περιοχής παρατήρησης και το L αντιπροσωπεύει την φασματική διάσταση (φασματικές ζώνες).



Εικόνα 1 - Υπερφασματικός κύβος [1]

Οι υπερφασματικοί κύβοι παράγονται από αερομεταφερόμενους αισθητήρες όπως ο AVIRIS της NASA, ή από δορυφόρους όπως ο EO-1 της NASA με το υπερφασματικό όργανο Hyperion. Ωστόσο, σε πολλές μελέτες ανάπτυξης χρησιμοποιούνται και αισθητήρες χειρός.

Από τεχνικής άποψης, υπάρχουν τέσσερις τρόποι με τους οποίους οι αισθητήρες παράγουν τον υπερφασματικό κύβο: Χωρική σάρωση, Φασματική σάρωση, Απεικόνιση στιγμιότυπου και Χωρο-φασματική σάρωση (Εικόνα 2). Η επιλογή του τρόπου εξαρτάται από την ιδιαιτερότητα κάθε εφαρμογής, καθώς κάθε τεχνική έχει πλεονεκτήματα και μειονεκτήματα που εξαρτώνται από το περιβάλλον παρατήρησης.



Εικόνα 2 - Τεχνικές σάρωσης υπερφασματικού κύβου [2]

Χωρική σάρωση

Στη χωρική σάρωση, κάθε έξοδος δύο διαστάσεων του αισθητήρα αντιπροσωπεύει ένα επίπεδο (x, L). Επομένως ο αισθητήρας σαρώνει την περιοχή ανά λωρίδα, έχοντας το μειονέκτημα ότι η εικόνα έχει συλλεχθεί μέσω της κίνησης μίας μηχανικής πλατφόρμας. Αυτό απαιτεί ακριβείς πληροφορίες τοποθεσίας ανά έξοδο, ώστε να επιτευχθεί αναδόμηση του υπερφασματικού κύβου. Παρ' όλα αυτά, τα συστήματα σάρωσης γραμμής είναι ιδιαίτερα συνηθισμένα στην απομακρυσμένη παρατήρηση, όπου χρησιμοποιούνται κινητές πλατφόρμες. Επιπλέον, τέτοιοι αισθητήρες χρησιμοποιούνται σε γραμμές παραγωγής για την σάρωση υλικών που κινούνται σε ένα μεταφορικό ιμάντα.

Φασματική σάρωση

Στην φασματική σάρωση, κάθε έξοδος δύο διαστάσεων του αισθητήρα αντιπροσωπεύει ένα χωρικό επίπεδο (x, y). Η διαδικασία σάρωσης βασίζεται σε οπτικά φίλτρα, καθώς η περιοχή σαρώνεται ανά φάσμα αλλάζοντας το ένα φίλτρο μετά το άλλο, ενώ η πλατφόρμα πρέπει να είναι ακίνητη. Σε τέτοιου είδους συστήματα ανίχνευσης μήκους κύματος, είναι πιθανό να προκύψει φασματική επικάλυψη εάν υπάρξει κίνηση εντός της σκηνής παρατήρησης, ακυρώνοντας όλο το υπερφασματικό κύβο δεδομένων. Παρ' όλα αυτά, υπάρχει το πλεονέκτημα ότι είναι εφικτή η άμεση παρατήρηση μίας συγκεκριμένης φασματικής ζώνης, έχοντας την αναπαράσταση της περιοχής σε χωρικές διαστάσεις.

Απεικόνιση στιγμιότυπου

Στην απεικόνιση στιγμιότυπου, κάθε έξοδος δύο διαστάσεων του αισθητήρα αντιπροσωπεύει όλα τα χωρικά (x, y) και φασματικά (L) δεδομένα. Οι αισθητήρες αποδίδουν ολόκληρο το υπερφασματικό κύβο ταυτόχρονα, χωρίς σάρωση. Ένα ενιαίο στιγμιότυπο απεικονίζει μία προοπτική προβολή του υπερφασματικού κύβου, από την οποία μπορεί να ανασυσταθεί η τρισδιάστατη δομή του. Βασικά πλεονεκτήματα αυτών των συστημάτων είναι η υψηλότερη απόδοση φωτός και ο μικρότερος χρόνος απόκτησης. Ωστόσο, η πολυπλοκότητα και το κόστος κατασκευής τους είναι υψηλό.

Χωρο-φασματική σάρωση

Στη χωρο-φασματική σάρωση, κάθε έξοδος δύο διαστάσεων του αισθητήρα αντιπροσωπεύει ένα επίπεδο (x, y). Το επίπεδο είναι χωρισμένο σε L λωρίδες κατά τον άξονα y, οι οποίες έχουν σαρωθεί στην αντίστοιχη φασματική ζώνη L. Προχωρημένα συστήματα χωρο-φασματικής σάρωσης μπορούν να προκύψουν τοποθετώντας ένα στοιχείο διασποράς (πρίσμα διασποράς) πίσω από ένα σύστημα χωρικής σάρωσης. Η σάρωση επιτυγχάνεται μέσω της κίνησης ολόκληρου του συστήματος σε σχέση με την σκηνή παρατήρησης. Η χωρο-φασματική σάρωση συνδυάζει μερικά από τα πλεονεκτήματα της χωρικής και της φασματικής σάρωσης, ελαχιστοποιώντας αρκετά από τα μειονεκτήματά τους.

2.3. Προ-Επεξεργασία Υπερφασματικών Δεδομένων

Η επεξεργασία των υπερφασματικών δεδομένων είναι περίπλοκη δεδομένου ότι περιέχει μεγάλο αριθμό φασματικών ζωνών που οδηγούν σε μεγάλο όγκο δεδομένων. Η υπερφασματική ποιότητα των δεδομένων επηρεάζεται από τα τυχαία σφάλματα του αισθητήρα και τον θόρυβο που προκαλείται στο σήμα. Αυτά τα σύνολα δεδομένων επηρεάζονται επίσης από ατμοσφαιρικά σφάλματα, επομένως απαιτείται μία διαδικασία προ-επεξεργασίας, που περιλαμβάνει την αφαίρεση των σφαλμάτων αισθητήρα και των ατμοσφαιρικών σφαλμάτων.

Διόρθωση σφάλματος αισθητήρα

Οι περισσότεροι από τους υπερφασματικούς αισθητήρες, όπως το εργαλείο Hyperion, είναι σαρωτές στους οποίους οι λανθασμένα βαθμονομημένοι ανιχνευτές παράγουν κακές κατακόρυφες γραμμές στην εικόνα. Λόγω της μη ιδανικής βαθμονόμησης, τα γειτονικά εικονοστοιχεία είτε έχουν σταθερές τιμές είτε χαμηλότερες μεταξύ τους. Αυτές οι λάθος γραμμές μπορούν να διορθωθούν αντικαθιστώντας τις ψηφιακές τιμές τους με τις μέσες τιμές των αριστερών και δεξιών γειτονικών τους εικονοστοιχείων.

Ατμοσφαιρική διόρθωση

Η ατμόσφαιρα διασκορπίζει ηλεκτρομαγνητική ενέργεια που στέλνεται από τον ήλιο στην επιφάνεια της Γης και από την επιφάνεια της Γης στον αισθητήρα. Επομένως, η ηλεκτρομαγνητική ενέργεια που λαμβάνει ο αισθητήρας μπορεί να είναι μεγαλύτερη ή μικρότερη από εκείνη που οφείλεται στην ανακλαστικότητα μόνο της επιφάνειας της γης. Οι προσπάθειες διόρθωσης της ατμόσφαιρας ελαχιστοποιούν αυτές τις επιπτώσεις στα φάσματα της εικόνας. Η ατμοσφαιρική διόρθωση θεωρείται παραδοσιακά απαραίτητη πριν από την ποσοτική ανάλυση υπερφασματικής εικόνας. Διάφοροι αλγόριθμοι διόρθωσης ατμοσφαιρικών σφαλμάτων έχουν αναπτυχθεί και στηριχθεί σε μεθόδους οι οποίες χωρίζονται σε δύο τύπους: σχετικές και απόλυτες[3].

Οι σχετικές μέθοδοι χωρίζονται σε τρεις τύπους: τη διόρθωση επίπεδου πεδίου, την εμπειρική διόρθωση γραμμής και την εσωτερική σχετική διόρθωση ανάκλασης.

Η απόλυτη ατμοσφαιρική διόρθωση βασίζεται σε μερικά μοντέλα ατμοσφαιρικής διόρθωσης που απαιτούν πληροφορίες σχετικά με την ατμοσφαιρική κατάσταση, το υψόμετρο, τη γεωμετρία μεταξύ ηλίου και δορυφόρου, την απορρόφηση νερού και το χρόνο απόκτησης της εικόνας. Οι μέθοδοι απόλυτης ατμοσφαιρικής διόρθωσης έχουν το πλεονέκτημα έναντι άλλων μεθόδων ότι μπορούν να λειτουργήσουν υπό οποιεσδήποτε ατμοσφαιρικές συνθήκες.

Κάποιες από αυτές είναι οι εξής:

FLAASH (Fast Line-of-sight Atmospheric Analysis of Spectral Hypercubes):

Το FLAASH είναι ένα εργαλείο μοντελοποίησης ατμοσφαιρικής διόρθωσης στο ENVI για ανάκτηση της φασματικής ανάκλασης από εικόνες υπερφασματικής ακτινοβολίας. Το FLAASH ενσωματώνει το μοντέλο μεταφοράς ακτινοβολίας MODTRAN 4 για την αντιστάθμιση των ατμοσφαιρικών σφαλμάτων.

ATCOR (Atmospheric and Topographic CORrection):

Ο αλγόριθμος ATCOR αναπτύχθηκε την τελευταία δεκαετία σε δύο διαφορετικούς τύπους (ATCOR 2 και ATCOR 3) που δημιουργήθηκαν από τον Dr.Richter του Γερμανικού Αεροδιαστημικού Κέντρου - DLR. Ο ATCOR 2 είναι ένας αλγόριθμος γρήγορου ρυθμού ατμοσφαιρικής διόρθωσης με εφαρμογή σε σχεδόν οριζόντια επίπεδα, ενώ ο ATCOR 3 έχει σχεδιαστεί για τραχιές τοπογραφικές επιφάνειες, χρησιμοποιώντας ένα ψηφιακό μοντέλο ανύψωσης (DEM) για την ατμοσφαιρική διόρθωση.

ATREM (Atmospheric REMoval Program):

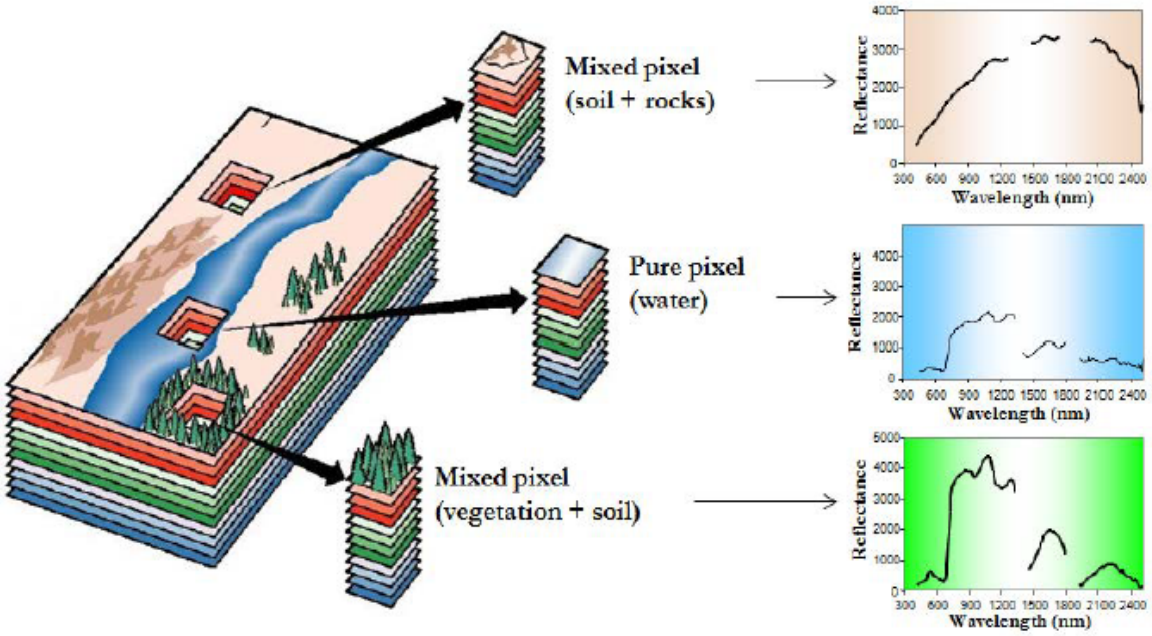
Το ATREM είναι ένα λογισμικό που αναπτύχθηκε από το Πανεπιστήμιο του Κολοράντο για την ανάκτηση της ανακλαστικότητας μίας επιφανείας από υπερφασματικά δεδομένα χρησιμοποιώντας ένα μοντέλο μεταφοράς ακτινοβολίας.

ACORN (Atmospheric CORrection Now):

Το πακέτο λογισμικού ACORN παρέχει μια ατμοσφαιρική διόρθωση δεδομένων υπερφασματικής και πολυφασματικής μέτρησης σε μήκη κύματος από 350 έως 2500 nm. Βασίζεται επίσης στο μοντέλο μεταφοράς ακτινοβολίας MODTRAN 4.

2.4. Μοντέλο Γραμμικού Φασματικού Διαχωρισμού

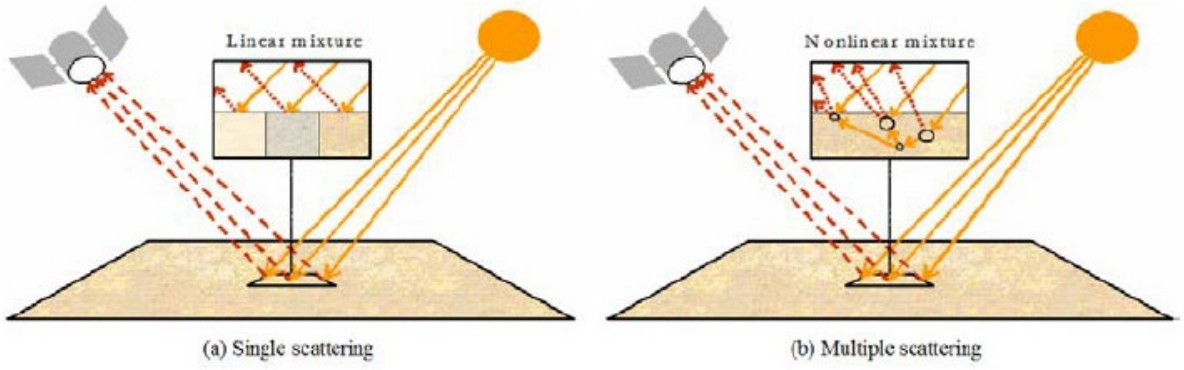
Ο υπερφασματικός διαχωρισμός είναι η διαδικασία εντοπισμού των endmembers και των ποσοστών αφθονίας σε μία υπερφασματική απεικόνιση. Με τον όρο endmember χαρακτηρίζεται η φασματική ανακλαστικότητα ενός καθαρού υλικού. Μια μέτρηση ανακλαστικότητας δεν μπορεί να είναι αρνητική, οπότε οι περισσότερες μέθοδοι διαχωρισμού περιορίζουν τις τιμές των endmembers να είναι μη αρνητικές. Σε μια αστική περιοχή τα endmembers μπορεί να είναι στέγες, άσφαλτος, δέντρα, γρασίδι, κ.λπ., ενώ μια περιοχή εξόρυξης μπορεί να έχει endmembers που αντιπροσωπεύουν διαφορετικούς τύπους μεταλλικών στοιχείων. Μια αφθονία εμφανίζεται συχνά ως ποσοστό, παρουσιάζοντας την σχετική περιοχή που καταλαμβάνει το endmember στο αντίστοιχο εικονοστοιχείο. Για παράδειγμα, το διάνυσμα εικονοστοιχείων που χαρακτηρίζεται ως "βλάστηση" στην παρακάτω εικόνα (Εικόνα 3) μπορεί στην πραγματικότητα να περιλαμβάνει ένα μείγμα βλάστησης και εδάφους. Σε αυτή την περίπτωση, αρκετά εικονοστοιχεία με καθαρές φασματικές υπογραφές(endmembers) συνδυάζονται στο ίδιο μικτό εικονοστοιχείο. Με τον όρο φασματική υπογραφή χαρακτηρίζεται η ακτινοβολία που αντανακλάται από μια επιφάνεια στα διάφορα μήκη κύματος (Εικόνα 3 δεξιά).



Εικόνα 3 - Μικτά εικονοστοιχεία [4]

Οι περισσότερες κλασσικές προσεγγίσεις για το διαχωρισμό έχουν βασιστεί στο μοντέλο γραμμικού φασματικού διαχωρισμού(linear spectral unmixing), που υποθέτει ότι τα

φάσματα των εικονοστοιχείων μπορούν να εκφραστούν με τη μορφή ενός γραμμικού συνδυασμού των φασματικών υπογραφών των endmembers που σταθμίζονται από τα αντίστοιχα ποσοστά αφθονίας(abundances). Αν και το γραμμικό μοντέλο έχει πρακτικά πλεονεκτήματα όπως η ευκολία εφαρμογής και η ευελιξία σε διαφορετικές εφαρμογές, μερικές φορές ο μη γραμμικός φασματικός διαχωρισμός μπορεί να χαρακτηρίσει καλύτερα το προκύπτον φασματικό μείγμα, ειδικά όταν τα endmembers κατανέμονται τυχαία σε όλο το οπτικό πεδίο του οργάνου. Στις περιπτώσεις αυτές, τα μικτά φάσματα συλλέγονται στο όργανο απεικόνισης υποθέτοντας ότι μέρος της ακτινοβολίας της πηγής είναι πολλαπλά σκεδασμένο πριν το συλλέξει ο αισθητήρας (Εικόνα 4).



Εικόνα 4 - Γραμμικό και Μη-γραμμικό μοντέλο [4]

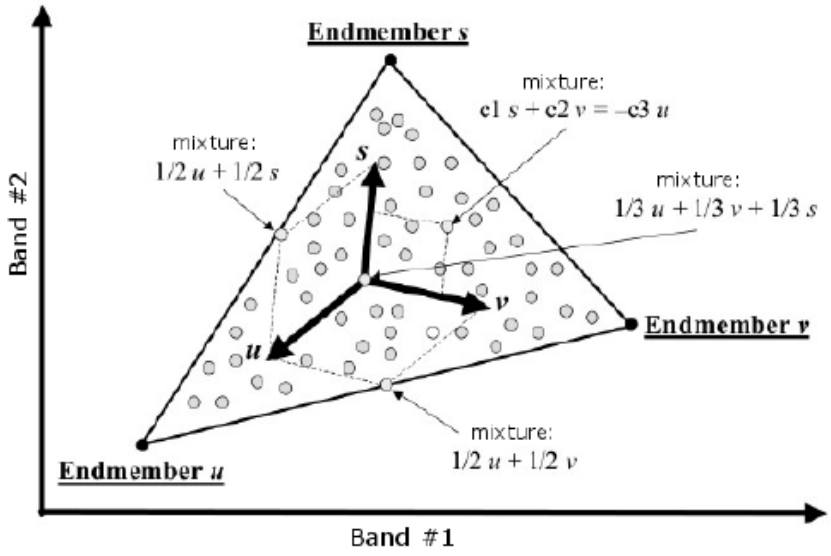
Ωστόσο ο γραμμικός φασματικός διαχωρισμός, παρά την απλότητά του, αποτελεί αποδεκτή προσέγγιση των μηχανισμών σκέδασης του φωτός σε πολλά πραγματικά σενάρια.

Έστω ότι $Y \in \mathbb{R}^{L \times n}$ είναι μια υπερφασματική εικόνα με L κανάλια και n εικονοστοιχεία. Ο πίνακας $Y = [y_1, \ldots, y_n]$ αντιπροσωπεύει μια υπερφασματική εικόνα σε μορφή μήτρας, όπου οι στήλες της είναι οι φασματικές υπογραφές y_i των εικονοστοιχείων της εικόνας και οι σειρές είναι οι φασματικές ζώνες της εικόνας. Σύμφωνα με την παραδοχή του μοντέλου γραμμικού διαχωρισμού τα υπερφασματικά δεδομένα μοντελοποιούνται ως εξής:

$$Y = MA + N$$

όπου $M \in \mathbb{R}^{L \times p}$, $M = [m_1, \ldots, m_p]$ πίνακας που περιέχει τα endmembers m_i ανά στήλη και $A \in \mathbb{R}^{p \times n}$ n, $A = [a1, \ldots, an]$ πίνακας που περιέχει τα abundances. Το άθροισμα των γινομένων όλων των abundances με τα αντίστοιχα endmembers αποτελεί την τιμή του κάθε εικονοστοιχείου. Τέλος, ο πίνακας $N \in \mathbb{R}^{L \times n}$ αντιπροσωπεύει το θόρυβο που εισάγεται στο μοντέλο κατά τη διαδικασία απεικόνισης[4].

Το μοντέλο του γραμμικού φασματικού διαχωρισμού μπορεί να ερμηνευτεί με γραφικό τρόπο χρησιμοποιώντας ένα διάγραμμα δυο διαστάσεων. Για παράδειγμα, στην Εικόνα 5 τα endmembers είναι τα πιο ακραία εικονοστοιχεία και ορίζουν ένα σχήμα(simplex) το οποίο περικλείει όλα τα άλλα εικονοστοιχεία. Κάθε εικονοστοιχείο μέσα στο simplex προκύπτει ως γραμμικός συνδυασμός των endmembers.

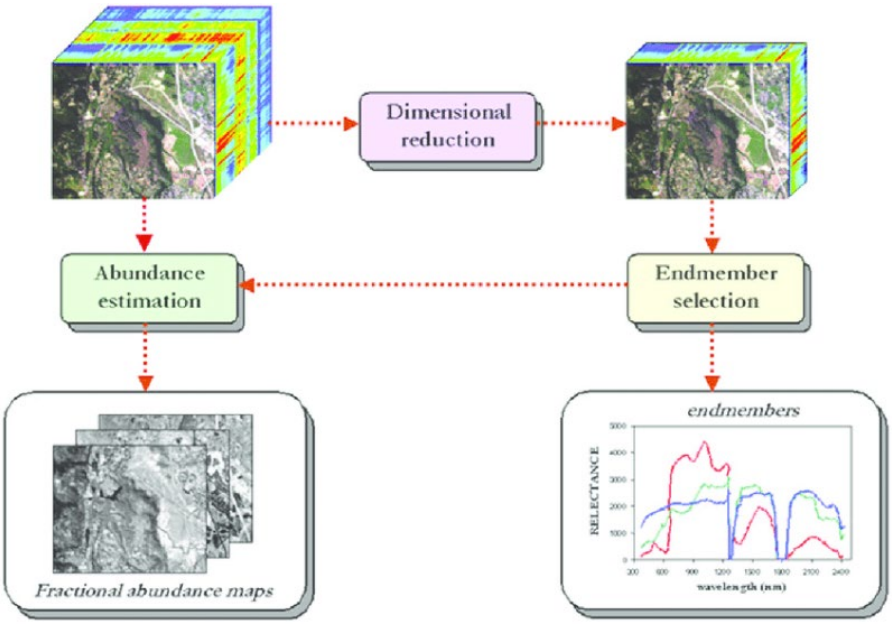


Εικόνα 5 - Διάγραμμα αναπαράστασης Γραμμικού μοντέλου [4]

2.5. Στάδια του Φασματικού Διαχωρισμού

Η λύση του προβλήματος του υπερφασματικού γραμμικού διαχωρισμού όπως περιγράφεται παραπάνω βασίζεται στην επιτυχή εκτίμηση του αριθμού των endmembers p, τα οποία είναι παρόντα σε μία υπερφασματική σκηνή Y, και στη σωστή εύρεση της ομάδας M των p endmembers και των αντίστοιχων ποσοστών τους σε κάθε εικονοστοιχείο, abundances.

Στο πρώτο βήμα, γίνεται μια προαιρετική μείωση των διαστάσεων των δεδομένων(Dimensionality reduction). Αυτό το βήμα είναι συνδεδεμένο με τον υπολογισμό των endmembers p, τα οποία υπάρχουν στην υπερφασματική σκηνή. Εφόσον ο αριθμός p βρεθεί, εφαρμόζεται το βήμα της συλλογής των endmembers, έτσι ώστε να ταυτοποιηθούν οι υπερφασματικές υπογραφές από τις οποίες απαρτίζεται η εικόνα(Endmember extraction). Τέλος, το βήμα της εύρεσης των abundances χρειάζεται ως είσοδο τις φασματικές υπογραφές που έχουν παραχθεί από πριν και δίνει σαν αποτέλεσμα μια ομάδα από χάρτες αφθονίας(abundance maps), οι οποίοι αντιστοιχούν σε κάθε φασματική υπογραφή(Abundance estimation)(Εικόνα 6).



Εικόνα 6 - Στάδια spectral unmixing [5]

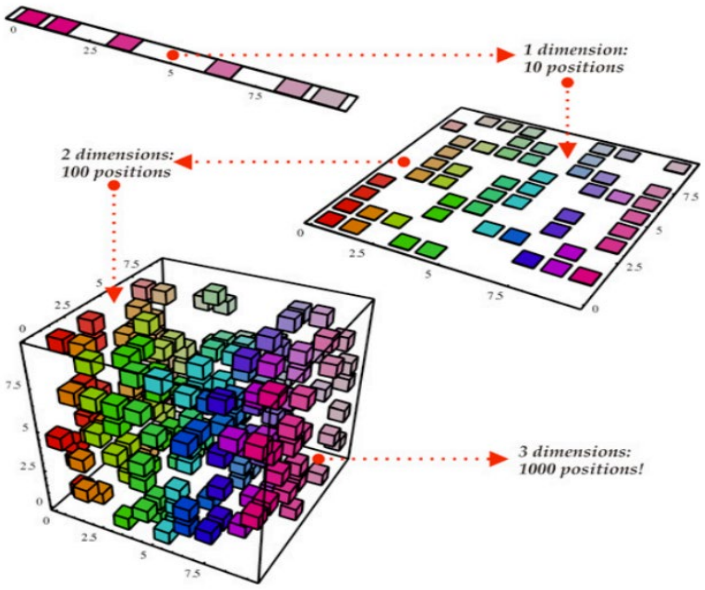
2.5.1. Dimensionality Reduction

Αν και τα υπερφασματικά δεδομένα είναι τόσο ογκώδη όσο και πολυδιάστατα, σήμερα με τη διαθεσιμότητα προηγμένων υπολογιστικών συστημάτων που διαθέτουν επεξεργαστές υψηλής ταχύτητας και τεράστια ισχύ αποθήκευσης, ο όγκος των δεδομένων δεν αποτελεί

πλέον περιορισμό. Το πρόβλημα έγκειται στον πλεονασμό των δεδομένων τα οποία πρέπει να μειωθούν ώστε να αποκτηθούν οι ζώνες με τη μέγιστη πληροφορία.

Τα πρόσφατα τεχνολογικά επιτεύγματα στη δορυφορική και αερομεταφερόμενη τηλεπισκόπηση έχουν αυξήσει τον όγκο των δεδομένων απομακρυσμένων φωτογραφιών που έχουν συλλεχθεί και βρίσκονται αποθηκευμένες σε βάσεις υπερφασματικών δεδομένων. Για παράδειγμα, ο ρυθμός απόκτησης δεδομένων του AVIRIS της NASA είναι 2.55 MB/s, δηλαδή μπορεί να συλλέξει έως και 9GB την ώρα. Το κινέζικο Pushbroom Hyperspectral Imager (PHI) έχει ρυθμό συλλογής δεδομένων 7.2 MB/s και μπορεί να συλλέξει περισσότερα από 25GB την ώρα, ενώ το υπερφασματικό όργανο Hyperion συλλέγει έναν υπερφασματικό κύβο με 256×6925 εικονοστοιχεία, 242 φασματικά κανάλια και ραδιομετρική ανάλυση της τάξεως των 12 bits ανά 30 sec, συλλέγοντας περίπου 71.9 GB την ώρα(πάνω από 1.6 TB ημερησίως)[6].

Η μείωση των διαστάσεων αναφέρεται στη διαδικασία μετατροπής δεδομένων με πολλές διαστάσεις σε δεδομένα τα οποία έχουν λιγότερες, εξασφαλίζοντας ότι διατηρούν την ίδια πληροφορία περιληπτικά. Για παράδειγμα, στην Εικόνα 7 φαίνεται πώς η μείωση των διαστάσεων ενός τρισδιάστατου κύβου επηρεάζει τον όγκο πληροφορίας.



Εικόνα 7 - Μείωση διαστάσεων κύβου πληροφορίας [7]

Τα δεδομένα που καταγράφονται από τους υπερφασματικούς αισθητήρες, έχουν συχνά σημαντική αλληλοεπικάλυψη του περιεχομένου της πληροφορίας μεταξύ των φασματικών ζωνών που καταγράφονται για ένα δεδομένο εικονοστοιχείο. Σε τέτοιες περιπτώσεις, δεν απαιτούνται όλα τα δεδομένα για να χαρακτηριστεί σωστά το εικονοστοιχείο. Ο αριθμός των endmembers p που υπάρχουν σε μια δεδομένη σκηνή είναι, συχνά, πολύ μικρότερος από τον αριθμό των ζωνών L. Επομένως, υποθέτοντας ότι το γραμμικό μοντέλο είναι καλό,

τα φασματικά διανύσματα βρίσκονται ή είναι πολύ κοντά σε ένα γραμμικό υποσύνολο χαμηλών διαστάσεων.

Η αναγνώριση αυτού του υποσυνόλου επιτρέπει την ακριβή αναπαράσταση των φασματικών διανυσμάτων σε χαμηλές διαστάσεις. Η επεξεργασία των δεδομένων τα οποία αντιπροσωπεύονται σε ένα υποσύστημα σημάτων είναι συνήθως συμφέρουσα και μερικές φορές απαραίτητη. Επομένως, ένας αλγόριθμος προσδιορισμού υποχώρου απαιτείται συχνά ως πρώτο βήμα στη διαδικασία φασματικού διαχωρισμού. Ο προσδιορισμός του υποχώρου χωρίς επίβλεψη έχει προσεγγιστεί με πολλούς τρόπους. Οι τεχνικές προβολής επιδιώκουν την εύρεση των καλύτερων υποσυστημάτων για την αναπαράσταση των δεδομένων βελτιστοποιώντας τις αντικειμενικές λειτουργίες. Για παράδειγμα, ο PCA(Principal Component Analysis) μεγιστοποιεί τη διακύμανση του σήματος, ο SVD(Singular Value Decomposition) μεγιστοποιεί την ισχύ και οι MNF(Minimum Noise Fraction) και NAPC(Noise-adjusted Principal Components) ελαχιστοποιούν το λόγο της ισχύος θορύβου προς την ισχύ σήματος. Ο NAPC είναι μαθηματικά ισοδύναμος με τον MNF και μπορεί να ερμηνευτεί ως μια ακολουθία δύο μετασχηματισμών βασικών συστατικών: ο πρώτος αφορά τον θόρυβο και ο δεύτερος αφορά το μετασχηματισμένο σύνολο δεδομένων[4].

2.5.2. Endmember Extraction

Ο φασματικός διαχωρισμός είναι πολύ σημαντικός για την ανάλυση και ταξινόμηση των υπερφασματικών εικόνων. Στόχος του είναι να προσδιορίσει ένα σύνολο από καθαρές υπογραφές αναφοράς(endmembers) και να καθορίσει τα ποσοστά αφθονίας τους για κάθε εικονοστοιχείο. Η ακρίβεια των ποσοστών αφθονίας εξαρτάται έντονα από την ακρίβεια των εντοπισμένων endmembers. Οι αλγόριθμοι εξαγωγής endmember(EEAs - Endmember Extraction Algorithms) αντιμετωπίζουν το φασματικό πρόβλημα είτε με γεωμετρική προσέγγιση είτε με στατιστική προσέγγιση.

Γεωμετρική προσέγγιση

Οι αλγόριθμοι που είναι βασισμένοι στη γεωμετρική προσέγγιση υποθέτουν ότι τα μετρούμενα φάσματα μπορούν να εκφραστούν ως ένας γραμμικός συνδυασμός των φασματικών υπογραφών των υλικών που παρουσιάζονται στο μικτό εικονοστοιχείο. Οι αλγόριθμοι αυτοί μπορούν να χωριστούν σε δυο κατηγορίες: προσεγγίσεις καθαρών εικονοστοιχείων(Pure Pixel - PP) και προσεγγίσεις ελαχίστου όγκου(Minimum Volume - MV).

Προσεγγίσεις καθαρού εικονοστοιχείου

Πολλές μέθοδοι έχουν κάνει την παραδοχή του PP, η οποία υποθέτει ότι κάθε υλικό στην εικόνα αντιπροσωπεύεται από τουλάχιστον ένα PP. Αυτή η υπόθεση διευκολύνει το σχεδιασμό των πολύ αποδοτικών υπολογιστικών μεθόδων, καθώς η εκπροσώπηση κάθε endmember από τουλάχιστον ένα αντιπροσωπευτικό εικονοστοιχείο των δεδομένων, μπορεί να κάνει τη διαδικασία επικύρωσης πιο απλή. Ακόμη, μπορεί να είναι ευκολότερο για τους επαγγελματίες του χώρου να ερμηνεύουν τα endmembers εάν αυτά είναι κατασκευασμένα από πραγματικά δεδομένα.

Πολύ γνωστές μέθοδοι που χρησιμοποιούν αυτή την προσέγγιση είναι, μεταξύ άλλων, ο PPI(Pixel Purity Index), ο N-FINDR και ο VCA(Vertex Component Analysis). Ο PPI ξεκινά με τη μείωση των διαστάσεων των δεδομένων χρησιμοποιώντας το μέγιστο κλάσμα θορύβου (Maximum Noise Fraction - MNF). Ένα μεγάλο σύνολο τυχαίων διανυσμάτων καθορίζεται, και στη συνέχεια κάθε φασματικό διάνυσμα των δεδομένων προβάλλεται επάνω σε αυτά τα διανύσματα. Μετά από αυτό, τα εικονοστοιχεία βαθμολογούνται και αυτά με την υψηλότερη βαθμολογία ορίζονται ως τα καθαρότερα εικονοστοιχεία στα δεδομένα. Ο N-FINDR δημιουργεί ένα simplex μέσα στα δεδομένα και βρίσκει το σύνολο των εικονοστοιχείων που μεγιστοποιούν τον όγκο αυτού του simplex. Ο VCA είναι μια επαναληπτική διαδικασία που χρησιμοποιεί τον υποχώρο που είναι ήδη καθορισμένος από τα endmembers και προβάλλει τα δεδομένα σε μια διεύθυνση ορθογώνια σε αυτόν. Το άκρο αυτής της προβολής (το εικονοστοιχείο που αντιστοιχεί στη μέγιστη τιμή) αντιστοιχεί στο νέο endmember. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν τα endmembers.

Προσεγγίσεις ελαχίστου όγκου

Αυτή η προσέγγιση δεν προϋποθέτει ότι τα endmembers αντιπροσωπεύονται από καθαρά εικονοστοιχεία στα δεδομένα, γεγονός που καθιστά τη βελτιστοποίηση μη κυρτή(Non-convex Optimization), και έτσι είναι πολύ πιο δύσκολο να λυθεί. Αυτή η προσέγγιση προσπαθεί να βρει ένα σύνολο από endmembers που ελαχιστοποιούν τον όγκο του simplex που ορίζουν, με τον περιορισμό ότι τα παρατηρούμενα δεδομένα βρίσκονται μέσα στο simplex. Αυτός ο περιορισμός μπορεί να είναι χαλαρός ή αυστηρός.

Τρεις γνωστές μέθοδοι που χρησιμοποιούν την προσέγγιση ελάχιστου όγκου είναι ο MSVA(Minimum Volume Simplex Analysis), ο SISAL(Simplex Identification via variable Splitting and augmented Lagrangian) και ο MVC-NMF(Minimum Volume transform-Nonnegative Matrix Factorization). Οι MSVA και SISAL επιτρέπουν την παραβίαση του περιορισμού του θετικού πρόσημου των δεδομένων έτσι ώστε να είναι σε θέση να αποκτήσουν ένα ορισμένο βαθμό ανθεκτικότητας. Ο MVC-NMF δεν παραβιάζει τον περιορισμό αυτό και η ελαχιστοποίηση γίνεται με εναλλακτική εκτίμηση των ποσοστών

αφθονίας και των endmembers με επαναληπτικό τρόπο. Επιπλέον ο MVC-NMF δεν μειώνει τις διαστάσεις των δεδομένων σε αντίθεση με το SISAL και το MSVA[8].

Στατιστική προσέγγιση

Οι αλγόριθμοι εξαγωγής endmember που βασίζονται στη στατιστική προσέγγιση υποθέτουν ότι τα στοιχεία κατανέμονται τυχαία στην εικόνα. Στις περιπτώσεις που υπάρχουν πολλά endmembers, ο αριθμός των φασματικών διανυσμάτων στις πλευρές του simplex μειώνεται και αυτό θα προκαλέσει χειρότερα αποτελέσματα σε σχέση με τις γεωμετρικές μεθόδους. Σε αυτά τα σενάρια, οι ερευνητές εξέτασαν τις στατιστικές προσεγγίσεις ως εναλλακτικές λύσεις. Αυτές οι προσεγγίσεις είναι πολύ ισχυρές, αλλά η υπολογιστική πολυπλοκότητα τους είναι υψηλότερη από την πολυπλοκότητα των γεωμετρικών. Χρησιμοποιώντας το στατιστικό πλαίσιο, ο φασματικός διαχωρισμός μπορεί να μετατραπεί σε ένα πρόβλημα στατιστικής επίτευξης και οι ιδιότητες της υποκείμενης κατανομής μπορούν να συναχθούν από την ανάλυση των δεδομένων.

Όταν είναι αναγκαίο να εκτιμηθούν τόσο τα ποσοστά αφθονίας όσο και τα endmembers, ο φασματικός διαχωρισμός είναι ένα πρόβλημα BSS(Blind Source Separation). Για την επίλυση προβλημάτων BSS χρησιμοποιούνται μέθοδοι όπως η ανάλυση ανεξάρτητων στοιχείων (Independent Component Analysis - ICA) και η NMF.

Οι Bayesian προσεγγίσεις είναι μεταξύ των στατιστικών προσεγγίσεων που έχουν εφαρμοστεί στο unmixing. Αυτές οι μέθοδοι αντιμετωπίζουν τα endmembers ως κατανομές και έτσι είναι ικανές να αιτιολογήσουν τη φασματική μεταβλητότητα. Ωστόσο, μπορεί να μην είναι διαθέσιμες στην πράξη οι ακριβείς γνώσεις σχετικά με τις κατανομές. Όταν συμβαίνει αυτό, οι άγνωστες κατανομές και οι παράμετροι, μαζί με τα ποσοστά αφθονίας εκτιμώνται από κοινού κατά τη διάρκεια του unmixing. Μια εξέχουσα στατιστική κατανομή που χρησιμοποιείται για τα endmembers είναι η NCM(Normal Composition Model), ενώ πολλές φορές χρησιμοποιείται μια δειγματοληπτική προσέγγιση της αλυσίδας Markov Monte Carlo, όπου οι μέσες τιμές των endmembers θεωρούνται γνωστές. Άλλες κατανομές που έχουν επίσης χρησιμοποιηθεί στον φασματικό διαχωρισμό είναι η BCM(Beta Composition Model) και η μέθοδος ροπών[8].

2.5.3. Abundance Estimation

Μετά την εξαγωγή των φασματικών υπογραφών, μπορούν να υπολογιστούν τα διανύσματα αφθονίας καθώς και οι χάρτες αφθονίας με τη χρήση του υπολογισμού των ελαχίστων τετραγώνων στο γραμμικό μοντέλο που αναλύθηκε προηγουμένως:

$$Y = MA + N$$

Με τον όρο διάνυσμα αφθονίας γίνεται αναφορά στην ποσότητα του κάθε υλικού που υπάρχει σε ένα εικονοστοιχείο, ενώ με τον όρο χάρτης αφθονίας γίνεται αναφορά στην ποσότητα ενός συγκεκριμένου υλικού που υπάρχει σε όλα τα εικονοστοιχεία. Η λύση της παραπάνω εξίσωσης δίνει διαφορετικές τιμές ανάλογα με τους περιορισμούς οι οποίοι εφαρμόζονται στα ποσοστά αφθονίας[9].

Εάν δεν υπάρχουν περιορισμοί (Unconstrained Least Squares - UCLS) το A είναι αυτό για το οποίο ελαχιστοποιείται το σφάλμα ανακατασκευής:

$$e = \|Y - MA\|^2$$

Η λύση σε αυτή την περίπτωση είναι:

$$A = (M^T M)^{-1} M^T Y$$

Στην περίπτωση όπου τα ποσοστά αφθονίας πρέπει να είναι θετικά (Nonnegativity Constrained Least Squares - NCLS), το πρόβλημα υπολογισμού των ποσοστών αφθονίας γίνεται ένα πρόβλημα περιορισμένης βελτιστοποίησης με ελάχιστο σφάλμα:

$$\min e = f(A) = Y^T Y - 2Y^T MA + A^T M^T M$$

$$0 \le a_i \le 1, με\ 1 \le i \le p$$

Το πρόβλημα αυτό λύνεται με τετραγωνικό προγραμματισμό, αφού η γραμμική συνάρτηση είναι τετραγωνική.

Τέλος, στην περίπτωση όπου τα ποσοστά αφθονίας πρέπει να είναι θετικά και να έχουν άθροισμα ίσο με ένα (Fully Constrained Least Squares - FCLS) το πρόβλημα περιορισμένης βελτιστοποίησης με ελάχιστο σφάλμα:

$$\min e = f(A) = Y^T Y - 2Y^T MA + A^T M^T M$$

$$0 \le a_i \le 1,\ a_1 + a_2 + \cdots + a_p = 1, με\ 1 \le i \le p$$

μπορεί να λυθεί με την αναγωγή του σε ένα πρόβλημα απαλλαγμένο από τον περιορισμό του αθροίσματος. Αυτή η διαδικασία γίνεται προσθέτοντας ένα διάνυσμα, με όλα τα στοιχεία του ίσα με ένα, στην τελευταία γραμμή των πινάκων M, Y και λύνοντας το πρόβλημα ελαχίστων τετραγώνων με τους αλλαγμένους πίνακες $\tilde{M},\ \tilde{Y}$ όπου:

$$\tilde{M} = \begin{bmatrix} M \\ 1 \end{bmatrix}, \qquad \tilde{Y} = \begin{bmatrix} Y \\ 1 \end{bmatrix}$$

ΚΕΦΑΛΑΙΟ 3 – APACHE SPARK

3.1. Επισκόπηση

Το Apache Spark είναι ένα εργαλείο ανάλυσης μεγάλων δεδομένων(Big Data) και μηχανικής μάθησης(Machine Learning). Αρχικά αναπτύχθηκε στο Πανεπιστήμιο του Μπέρκλεϋ το 2009 και πλέον θεωρείται το μεγαλύτερο έργο ανοιχτού κώδικα στην επεξεργασία δεδομένων. Από την κυκλοφορία του έως τώρα, αρκετές εταιρίες έχουν ενσωματώσει το Apache Spark σε ένα ευρύ φάσμα εφαρμογών τους. Για παράδειγμα το Netflix, το Yahoo και το eBay έχουν αξιοποιήσει το Spark σε μαζική κλίμακα, επεξεργάζοντας πολλαπλά PetaBytes δεδομένων σε υπολογιστικές συστάδες(clusters) άνω των 8.000 κόμβων. Η κοινότητα του Spark έχει μετατραπεί σε μικρό χρονικό διάστημα στη μεγαλύτερη κοινότητα ανοιχτού κώδικα στον κλάδο των Big Data, με περισσότερους από 1000 συμμετέχοντες και περισσότερους από 250 οργανισμούς.

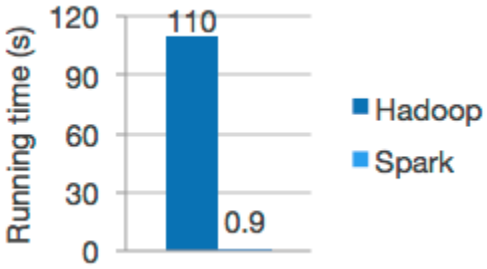
Ιστορικά, το Spark, ως συνέχεια του Hadoop, αναπτύχθηκε το 2009 στο εργαστήριο AMPLab του Πανεπιστήμιο του Μπέρκλεϋ από τον Matei Zaharia. Ήταν λογισμικό ανοιχτού κώδικα(Open Source) έως το 2010 με άδεια BSD(Berkeley Software Distribution). Το 2013 χορηγήθηκε στο ίδρυμα λογισμικού Apache(Apache software foundation) και από τον Φεβρουάριο του 2014 θεωρείται ένα από τα κορυφαία έργα της Apache[10].

Στη βιομηχανία χρησιμοποιείται εκτεταμένα το Hadoop για την ανάλυση συνόλων δεδομένων. Ο λόγος είναι ότι το πλαίσιό του βασίζεται σε ένα απλό μοντέλο προγραμματισμού, το MapReduce, που επιτρέπει μια υπολογιστική λύση επεκτάσιμη, ευέλικτη, ανεκτική σε σφάλματα και οικονομικά αποδοτική. Στην περίπτωση του Spark, η κύρια ανησυχία είναι η διατήρηση της ταχύτητας επεξεργασίας μεγάλων συνόλων δεδομένων μειώνοντας τόσο το χρόνο αναμονής μεταξύ των ερωτημάτων όσο και το χρόνο αναμονής για την εκτέλεση του προγράμματος.

Το Spark παρουσιάστηκε από την Apache για την επιτάχυνση της υπολογιστικής διαδικασίας του Hadoop. Αντίθετα με την κοινή πεποίθηση, δεν αποτελεί μια τροποποιημένη έκδοση του Hadoop και δεν εξαρτάται πραγματικά από αυτό, καθώς έχει τη δική του διαχείριση των clusters. Το Spark μπορεί να χρησιμοποιήσει το Hadoop για δύο σκοπούς: ο ένας είναι η αποθήκευση δεδομένων και ο άλλος η επεξεργασία τους. Δεδομένου όμως ότι έχει δικό του σύστημα διαχείρισης υπολογισμών στο cluster, χρησιμοποιεί το Hadoop μόνο για σκοπούς αποθήκευσης.

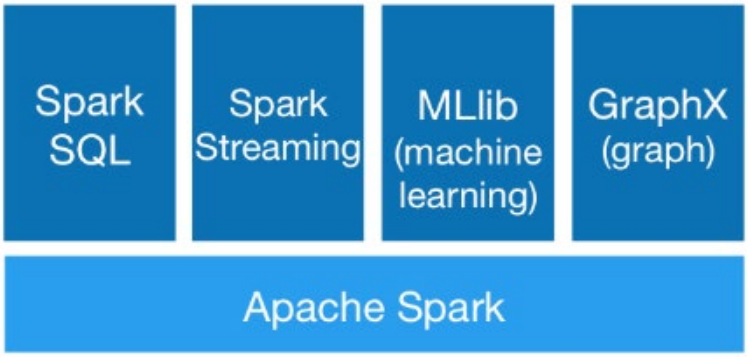
Το κύριο χαρακτηριστικό του Spark είναι ότι πραγματοποιεί τους υπολογισμούς του cluster εντός της μνήμης(in-memory cluster computing), γεγονός που αυξάνει την ταχύτητα επεξεργασίας μιας εφαρμογής. Σχεδιασμένο εξολοκλήρου για να είναι αποδοτικό, το Spark

μπορεί να είναι 100 φορές ταχύτερο από το Hadoop(Εικόνα 8). Αυτό επιτυγχάνεται με τη μείωση του αριθμού των εργασιών ανάγνωσης και εγγραφής στο δίσκο, καθώς αποθηκεύει τα ενδιάμεσα δεδομένα επεξεργασίας στη μνήμη. Παρόλα αυτά παραμένει γρήγορο ακόμη και όταν τα δεδομένα αποθηκεύονται στο δίσκο (έως και 10 φορές ταχύτερο από το Hadoop) και αυτή τη στιγμή κατέχει το παγκόσμιο ρεκόρ της ταξινόμησης δεδομένων μεγάλης κλίμακας σε δίσκο[11].



Εικόνα 8 - Σύγκριση ταχύτητας Hadoop-Spark [11]

Το Spark σχεδιάστηκε για να καλύψει ένα ευρύ φάσμα εργασιών, όπως εφαρμογές επεξεργασίας δέσμης(batch processing), επαναληπτικούς αλγόριθμους, διαδραστικά ερωτήματα και streaming. Υποστηρίζει πολλαπλές γλώσσες, καθώς παρέχει ενσωματωμένες διεπαφές προγραμματισμού εφαρμογών(Application Programming Interface – APIs) σε Java, Scala, Python και R. Επιπλέον, περιλαμβάνει μια συλλογή από περισσότερους από 100 operators για τη μετατροπή δεδομένων και πολλές γνωστές data frame APIs για τον χειρισμό ημιδομημένων δεδομένων. Το Spark συνοδεύεται από βιβλιοθήκες υψηλότερου επιπέδου, συμπεριλαμβανομένης της υποστήριξης ερωτημάτων SQL(Spark SQL), δεδομένων ροής(Spark Streaming), μηχανικής μάθησης(MLlib) και επεξεργασίας γραφημάτων(GraphX)(Εικόνα 9). Αυτές οι τυποποιημένες βιβλιοθήκες αυξάνουν την παραγωγικότητα και μπορούν να συνδυαστούν για τη δημιουργία πολύπλοκων ροών εργασίας.



Εικόνα 9 - Βιβλιοθήκες του Apache Spark [11]

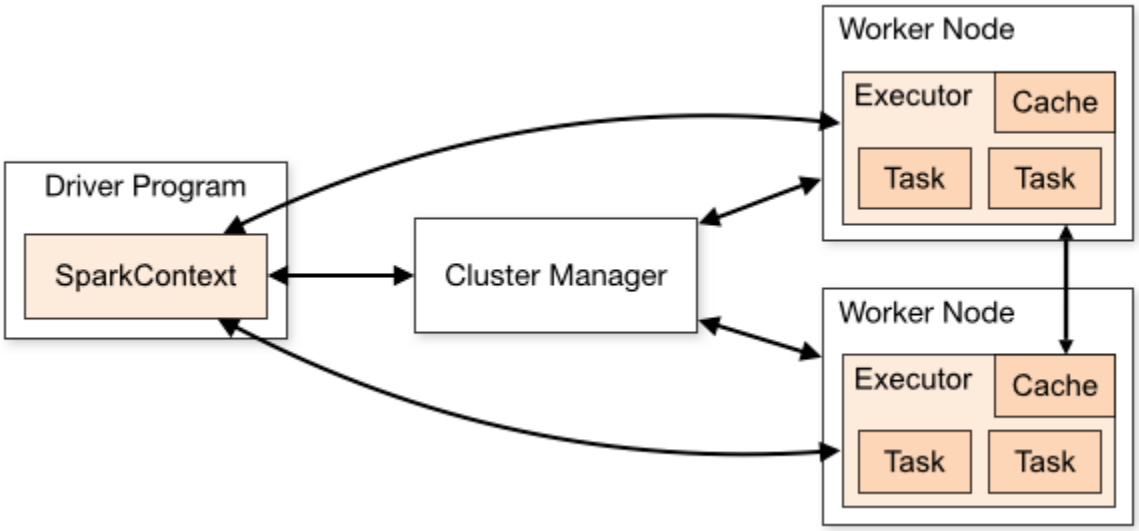
Το Apache Spark απαιτεί ένα διαχειριστή του cluster και ένα κατανεμημένο σύστημα αποθήκευσης δεδομένων. Για τη διαχείριση του cluster, υποστηρίζονται ο Standalone(τοπικός διαχειριστής του Spark), ο Hadoop YARN ή ο Apache Mesos. Σχετικά με την κατανεμημένη αποθήκευση, το Spark μπορεί να αλληλεπιδράσει με συστήματα όπως το HDFS(Hadoop Distributed File System), το MapR-FS(MapR File System), το Apache Cassandra, το Apache HBase, το Apache Hive, το OpenStack Swift, το Amazon S3 , το Kudu και άλλα(Εικόνα 10).

Εικόνα 10 - Οικοσύστημα Spark [11]

Τέλος, υποστηρίζεται μια ψευδο-κατανεμημένη τοπική λειτουργία, η οποία συνήθως χρησιμοποιείται μόνο για σκοπούς ανάπτυξης ή δοκιμής, όπου δεν απαιτείται κατανεμημένη αποθήκευση και αρκεί το τοπικό σύστημα αρχείων. Σε ένα τέτοιο σενάριο, το Spark τρέχει σε ένα μόνο μηχάνημα με έναν executor ανά πυρήνα CPU.

3.2. Αρχιτεκτονική Apache Spark

Το Spark χρησιμοποιεί μία αρχιτεκτονική αφέντη/σκλάβου(master/slave) σε ένα cluster μηχανημάτων. Το cluster αποτελείται από ένα κεντρικό μηχάνημα(master) και τουλάχιστον δύο ακόμη, τα οποία ονομάζονται workers. Υπάρχει ένας κεντρικός συντονιστής, το πρόγραμμα οδηγός(driver program), που επικοινωνεί με πολλούς κατανεμημένους κόμβους(executors), οι οποίοι ζουν στα μηχανήματα workers(Εικόνα 11).



Εικόνα 11 - Αρχιτεκτονική Apache Spark [11]

3.2.1. Driver Program

Ο driver είναι μία διεργασία η οποία περιέχει τη βασική συνάρτηση(main function) της εφαρμογής που τρέχει στο cluster και έχει πρόσβαση στο Spark μέσω του αντικειμένου(object) SparkContext, το οποίο αναπαριστά μία σύνδεση με το cluster. Η διεργασία αυτή τρέχει τον κώδικα του χρήστη, ο οποίος δημιουργεί το SparkContext, το βασικό τύπο δεδομένων του Spark(RDD) και εκτελεί όλα τα transformations και τα actions. Το driver πρόγραμμα ζει συνήθως στο μηχάνημα master και σηκώνει διάφορες κατανεμημένες διεργασίες σε ένα cluster. Παρόλα αυτά, δίνεται η δυνατότητα στο χρήστη να επιλέξει αν ο driver θα τρέξει σε ένα από τα μηχανήματα workers. Αυτό είναι συνετό στην περίπτωση που ο master είναι απομακρυσμένος από το υπόλοιπο cluster[12].

Το driver πρόγραμμα είναι υπεύθυνο για τη μετατροπή του κώδικα χρήστη σε μικρότερα καθήκοντα(tasks) και για τον προγραμματισμό της εκτέλεσής τους στους executors. Θεωρητικά, όλα τα προγράμματα στο Spark ακολουθούν την ίδια δομή: δημιουργούν RDDs από κάποια inputs, παράγουν νέα RDDs από αυτά χρησιμοποιώντας μετασχηματισμούς και πραγματοποιούν ενέργειες για τη συλλογή ή την αποθήκευση δεδομένων. Ένα πρόγραμμα Spark δημιουργεί ένα λογικό κατευθυνόμενο ακυκλικό γράφημα(DAG) των παραπάνω λειτουργιών του και ο driver, με την σειρά του, μετατρέπει αυτό το λογικό γράφημα σε ένα πλάνο εκτέλεσης χωρίζοντάς το σε tasks. Το Spark χρησιμοποιεί διάφορες βελτιστοποιήσεις όπως είναι η συγχώνευση των transformations, ώστε να μετατρέψει το DAG σε ένα σύνολο από στάδια(stages). Κάθε stage, με την σειρά του, αποτελείται από πολλά tasks. Επιπλέον ο driver πρέπει να συντονίσει τον προγραμματισμό εκτέλεσης των tasks στους executors, οι οποίοι κατά την εκκίνηση τους συνδέονται στον driver, έτσι ώστε να υπάρχει πλήρη παρακολούθησή τους από αυτόν. Κάθε executor, πέρα από το να εκτελεί tasks, αποθηκεύει δεδομένα RDD. Επομένως, ο driver αναθέτει το κάθε task στον executor, ο οποίος κατέχει το αντίστοιχο τμήμα των δεδομένων. Ακόμη, γνωρίζει κάθε στιγμή την θέση των δεδομένων που αποθηκεύονται στη μνήμη(cached data) έτσι ώστε να δρομολογήσει και τις μελλοντικές διαδικασίες πάνω σε αυτά.

Τέλος, ο driver αναλαμβάνει να παρουσιάσει πληροφορίες σχετικά με την τρέχουσα Spark εφαρμογή μέσω μίας διαδικτυακής διεπαφής χρήστη(web UI), η οποία είναι προσβάσιμη στην πόρτα 4040(http://masternode:4040).

3.2.2. Executor

Οι Spark executors είναι διεργασίες των workers υπεύθυνες για την εκτέλεση διαφορετικών tasks μίας Spark εργασίας(job). Οι executors δημιουργούνται κατά την εκτέλεση μίας Spark εφαρμογής και υπάρχουν καθόλη τη διάρκεια της, ενώ η ίδια η εφαρμογή μπορεί να συνεχίσει να εκτελείται ανεξάρτητα από το αν κάποιος executor αποτύχει. Αρχικά, ο πρώτος ρόλος τους είναι να αναλαμβάνουν τα tasks της εφαρμογής και να αποστέλλουν τα αποτελέσματα στον driver. Ενώ, ο δεύτερος να παρέχουν in-memory αποθήκευση για τα RDDs που αποθηκεύονται(cached) από τα προγράμματα χρήστη μέσω μίας υπηρεσίας, που ονομάζεται Block Manager και υπάρχει μέσα σε κάθε executor. Ο Block Manager διαχειρίζεται την αποθήκη των blocks τα οποία αντιπροσωπεύουν τμήματα των cached RDDs και παρέχει διεπαφές για την εγγραφή και ανάγνωσή τους, τόσο τοπικά όσο και απομακρυσμένα, σε διάφορα αποθηκευτικά συστήματα όπως μνήμη, δίσκος και άλλα[13].

3.2.3. Cluster Manager

Ο driver και οι executors τρέχουν ο καθένας σε μία δική του εικονική μηχανή της Java(Java Virtual Machine - JVM) και όλοι μαζί αποτελούν τον όρο Spark εφαρμογή. Μία Spark εφαρμογή εκτελείται σε ένα σύνολο μηχανημάτων χρησιμοποιώντας μία εξωτερική υπηρεσία, η οποία ονομάζεται cluster manager. Το Spark έχει έναν ενσωματωμένο cluster manager, ο οποίος ονομάζεται Standalone, ενώ μπορεί να δουλέψει ακόμη και με δύο άλλους γνωστούς open source cluster managers, τους Hadoop YARN και Apache Mesos. Η εκκίνηση των executors και, σε ορισμένες περιπτώσεις, του driver βασίζεται στον cluster manager. Ο driver ζητά executors από τον cluster manager και στη συνέχεια δρομολογεί σε αυτούς τα tasks της εφαρμογής. Τέλος, ο cluster manager είναι υπεύθυνος για τον προγραμματισμό και τη διανομή των πόρων ανάμεσα στα μηχανήματα του cluster.

Standalone Cluster Manager

Ο Standalone cluster manager του Spark προσφέρει έναν απλό τρόπο εκτέλεσης εφαρμογών σε ένα cluster, το οποίο αποτελείται από ένα master και πολλούς workers, με μια διαμορφωμένη ποσότητα μνήμης και αριθμό πυρήνων CPU. Όταν μία εφαρμογή υποβάλλεται για εκτέλεση, ο χρήστης μπορεί να επιλέξει τόσο το πόση μνήμη θα χρησιμοποιήσουν οι executors όσο και πόσους πυρήνες. Ο Standalone κατανέμει αυτόματα την ποσότητα της CPU και της μνήμης σε κάθε worker και προσφέρει μία διαδικτυακή διεπαφή χρήστη(web UI), η οποία είναι προσβάσιμη στην πόρτα 8080(http://masternode:8080). Μέσω αυτής ο χρήστης μπορεί να επιβεβαιώσει των αριθμό των πυρήνων της CPU, τη μνήμη που χρησιμοποιεί η εφαρμογή, καθώς και το χρόνο εκτέλεσης της.

Υποστηρίζονται δύο τρόποι ανάπτυξης(deploy modes) από τον Standalone για το πού τρέχει ο driver της εφαρμογής. Στη λειτουργία πελάτη(client mode),που είναι η προεπιλογή, ο driver τρέχει στο μηχάνημα όπου εκτελείται η εντολή εκκίνησης του Spark, spark-submit. Αυτό δίνει τη δυνατότητα στο χρήστη να παρακολουθεί την έξοδο του προγράμματος οδήγησης και να το τροφοδοτεί με δεδομένα εισόδου ανά πάσα στιγμή. Για το mode αυτό είναι απαραίτητο το μηχάνημα, από το οποίο έγινε η εκκίνηση, να έχει γρήγορη σύνδεση με τους workers και να παραμένει διαθέσιμο καθόλη τη διάρκεια της εφαρμογής. Αντίθετα, σε λειτουργία cluster(cluster mode), ο driver ξεκινάει ως μια άλλη διεργασία σε έναν από τους workers μέσα στο cluster και στη συνέχεια συνδέεται ξανά με τους executors. Σε αυτή τη λειτουργία το μηχάνημα από το οποίο έγινε η εκκίνηση δεν είναι αναγκαίο να μείνει ανοιχτό για όσο τρέχει η εφαρμογή. Ο χρήστης εξακολουθεί να έχει πρόσβαση στα αρχεία καταγραφής(log files) της εφαρμογής μέσω του web UI του Standalone. Η λειτουργία

cluster ενεργοποιείται κατά την εκκίνηση του Spark, προσθέτοντας την παράμετρο --deploy-mode στο spark-submit.

Στον Standalone η κατανομή των πόρων ελέγχεται από δύο ρυθμίσεις. Αρχικά, ο χρήστης μπορεί να διαμορφώσει τη μνήμη του executor χρησιμοποιώντας την παράμετρο --executor-memory στο spark-submit. Κάθε εφαρμογή έχει το πολύ έναν executor σε κάθε worker, για αυτό η ρύθμιση καθορίζει πόση από τη μνήμη του worker θα χρησιμοποιήσει η εφαρμογή(από προεπιλογή, αυτή η ρύθμιση είναι 1 GB). Ενώ, ο συνολικός αριθμός των πυρήνων που χρησιμοποιούνται μεταξύ των executors σε μια εφαρμογή(από προεπιλογή, είναι απεριόριστος) χρειάζεται, αν υπάρχουν πολλοί χρήστες, να οριστεί με βάση το φόρτο των διαφόρων εφαρμογών. Ο χρήστης μπορεί να διαμορφώσει αυτή την ρύθμιση μέσω της παραμέτρου --total-executor-cores κατά το spark-submit ή μέσω του Spark configuration file ορίζοντας το spark.cores.max[14].

Τέλος, ο Standalone cluster manager λειτουργεί, από προεπιλογή, με την διάδοση-διαμοιρασμό κάθε εφαρμογής μεταξύ του μέγιστου αριθμού executors. Για παράδειγμα, υποθέτοντας ότι το cluster αποτελείται από 20 κόμβους με μηχανήματα 4 πυρήνων και η εκκίνηση γίνεται με --executor-memory 1G και --total-executor-cores 8, το Spark θα εκκινήσει 8 executors, έκαστος με 1 GB μνήμης RAM, σε διαφορετικά μηχανήματα. Αυτό γίνεται από προεπιλογή για να δοθεί στις εφαρμογές η ευκαιρία να επιτύχουν data locality για τα κατανεμημένα συστήματα αρχείων που εκτελούνται στα ίδια μηχανήματα (π.χ. HDFS), επειδή αυτά τα συστήματα συνήθως έχουν δεδομένα που απλώνονται σε όλους τους κόμβους. Ο χρήστης όμως έχει τη δυνατότητα να ζητήσει από το Spark να ενοποιήσει τους executors σε όσο το δυνατόν λιγότερους κόμβους, δηλαδή, στην προκειμένη περίπτωση, να αυξήσει τον αριθμό των cores που χρησιμοποιεί κάθε executor. Αυτό ρυθμίζεται αλλάζοντας την τιμή της ιδιότητας spark.deploy.spreadOut σε ψευδή στο αρχείο conf/spark-defaults.conf. Στην περίπτωση αυτή, η προηγούμενη εφαρμογή θα είχε μόνο δύο executors, με 1 GB RAM και 4 πυρήνες ο καθένας. Αυτή η ρύθμιση επηρεάζει όλες τις εφαρμογές του Standalone και πρέπει να ρυθμιστεί πριν γίνει η εκκίνηση του cluster manager.

Hadoop YARN

Ο YARN είναι ένας cluster manager, που παρουσιάστηκε στο Hadoop 2.0, ο οποίος επιτρέπει σε διάφορα data processing frameworks να εκτελούνται σε ένα cluster κοινών πόρων και συνήθως είναι εγκατεστημένος στους ίδιους κόμβους με το HDFS. Η χρήση του YARN διευκολύνει το Spark, καθώς κάνει την πρόσβαση στα HDFS δεδομένα πιο γρήγορη. Αρχικά ορίζεται μια μεταβλητή περιβάλλοντος που δείχνει στον κατάλογο ρυθμίσεων του Hadoop(Hadoop configuration directory) και, στη συνέχεια, οι εργασίες υποβάλλονται σε ένα ειδικό master URL με το spark-submit[14].

Όπως και με τον Standalone, υπάρχουν δύο modes με τα οποία η εφαρμογή συνδέεται στο cluster. Στο client mode ο driver της εφαρμογής τρέχει στο μηχάνημα από το οποίο γίνεται το submit, ενώ στο cluster mode τρέχει μέσα σε ένα YARN container. Ο χρήστης διαμορφώνει το mode με την παράμετρο –deploy-mode στο spark-submit.

Όταν εκτελείται ο YARN, το Spark χρησιμοποιεί έναν σταθερό αριθμό από executors, ο οποίος ορίζεται μέσω της παραμέτρου –num-executors και από προεπιλογή είναι ίσος με δυο. Ο χρήστης έχει τη δυνατότητα ακόμη να ρυθμίσει τη μνήμη κάθε executor(--executor-memory) και τον αριθμό των πυρήνων που χρησιμοποιεί ο YARN(--executor-cores). Το Spark συνήθως τρέχει καλύτερα με λίγους και μεγαλύτερους executors, πολλών πυρήνων και μεγάλης μνήμης, καθώς έτσι μπορεί να βελτιώσει την επικοινωνία εντός του executor. Αξίζει να σημειωθεί ότι μερικά clusters έχουν μέγιστο όριο μνήμης για κάθε executor τα 8GB.

Τέλος, μερικά YARN clusters έχουν ρυθμιστεί ώστε να δρομολογούν τις εφαρμογές σε πολλαπλές ουρές για λόγους σωστότερης διαχείρισης πόρων. Αυτό είναι εφικτό μέσω της παραμέτρου --queue.

Apache Mesos

Ο Apache Mesos είναι ένας cluster manager γενικής χρήσης που μπορεί να τρέξει τόσο βαριές διεργασίες στατιστικών δεδομένων όσο και διεργασίας με μεγάλη διάρκεια. Σε αντίθεση με τους άλλους cluster managers, ο Mesos προσφέρει δύο modes για το διαμοιρασμό πόρων μεταξύ των executors στο ίδιο cluster. Στο fine-grained mode, που είναι το προεπιλεγμένο, οι executors επιλέγουν τον αριθμό των πυρήνων που χρησιμοποιούν δυναμικά κατά το χρόνο εκτέλεσης των tasks. Στο coarse-grained mode, το Spark διαθέτει συγκεκριμένο αριθμό πυρήνων σε κάθε executor και τους δεσμεύει καθόλη τη διάρκεια της εφαρμογής. Αυτό ρυθμίζεται μέσω της παραμέτρου --conf spark.mesos.coarse=true στο spark-submit[14].

Το fine-grained mode του Mesos είναι χρήσιμο όταν πολλοί χρήστες μοιράζονται ένα cluster για να τρέξουν διαδραστικές εφαρμογές όπως shells, όπου θα υπάρχουν εφαρμογές που θα μειώνουν τον αριθμό των πυρήνων που χρησιμοποιούν όταν είναι αδρανείς και θα επιτρέπουν σε άλλες πιο βαριές να κάνουν χρήση του cluster. Το μειονέκτημα, όμως, είναι ότι αυξάνεται η αναμονή ανάμεσα στα tasks, με αποτέλεσμα εφαρμογές χαμηλής-αναμονής(low-latency) να δυσκολεύονται, αλλά και άλλου είδους εφαρμογές να καθυστερούν περιμένοντας τους απαραίτητους πυρήνες για να τρέξουν. Παρόλα αυτά, είναι εντυπωσιακό ότι μπορούν να υπάρχουν εφαρμογές στο ίδιο cluster που χρησιμοποιούν διαφορετικά modes.

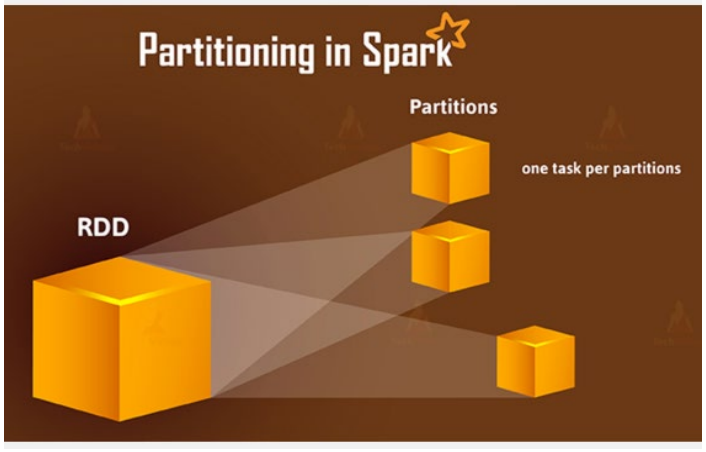
Το Spark με τον Mesos υποστηρίζει την εκτέλεση εφαρμογών μόνο στο client mode, δηλαδή ο driver μπορεί να τρέχει μόνο στο μηχάνημα που έγινε το submit. Εάν ο χρήστης θελήσει να τρέξει τον driver αλλού μέσα στο cluster θα χρησιμοποιήσει frameworks όπως το Aurora ή το Chronos που επιτρέπουν τον αναπρογραμματισμό των διεργασιών σε άλλα μηχανήματα, σε περίπτωση αποτυχίας του μηχανήματος στο οποίο τρέχει ο driver.

Τέλος, δίνεται η δυνατότητα στο χρήστη να ρυθμίσει τη μνήμη κάθε executor(--executor-memory) και το μέγιστο αριθμό πυρήνων της εφαρμογής (--total-executor-cores). Από προεπιλογή, το Spark δίνει στον executor όσους παραπάνω πυρήνες μπορεί, καθώς προσέχει κάθε εφαρμογή να τρέχει στον ελάχιστο δυνατό αριθμό από executors(αν δεν οριστεί η παράμετρος –total-executor-cores, θα προσπαθήσει να χρησιμοποιήσει όλους τους διαθέσιμους πυρήνες του cluster).

3.3. Πυρήνας Apache Spark

3.3.1 Resilient Distributed Dataset – RDD

Η βασική δομή του Spark είναι το Resilient Distributed Dataset (RDD), το οποίο είναι μία κατανεμημένη αμετάβλητη συλλογή αντικειμένων. Όλες οι διεργασίες εκφράζονται είτε ως δημιουργία RDD, είτε ως μετασχηματισμός ενός υπάρχοντος RDD είτε ως κλήση διάφορων operations σε RDDs για να υπολογιστεί κάποιο αποτέλεσμα. Το RDD έχει σχεδιαστεί με τέτοιο τρόπο ώστε να κρύβει μεγάλο μέρος της υπολογιστικής πολυπλοκότητας από το χρήστη, καθώς το Spark αυτόματα παραλληλοποιεί μέσα στο cluster τα δεδομένα που περιλαμβάνουν τα RDDs. Η βασική μονάδα παραλληλισμού των RDDs είναι το partition, το οποίο αποτελεί μία λογική διαίρεση των δεδομένων. Κάθε RDD χωρίζεται σε πολλά partitions και κάθε ένα από αυτά αντιστοιχεί σε ένα task, το οποίο αναλαμβάνει να υπολογίσει ένας κόμβος του cluster(Εικόνα 12). Ο αριθμός των partitions μπορεί να ρυθμιστεί από το χρήστη, αλλά συνήθως δεν πρέπει να είναι μικρότερος από τον συνολικό αριθμό των πυρήνων του cluster, καθώς αυτό θα σήμαινε ότι δε χρησιμοποιούνται όλοι οι πόροι στο μέγιστο. Όταν χρησιμοποιείται κατανεμημένο σύστημα αρχείων HDFS, ο προεπιλεγμένος αριθμός των partitions ισούται με το πλήθος των HDFS partitions(κάθε HDFS partition είναι 64MB)[13].



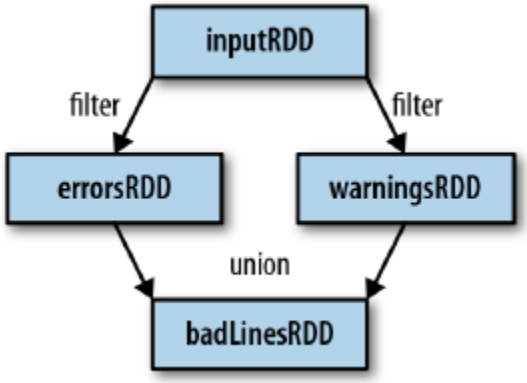
Εικόνα 12 - Παραλληλισμός στο Spark [15]

Ο χρήστης μπορεί να δημιουργήσει RDD είτε φορτώνοντας ένα εξωτερικό σύνολο δεδομένων στο driver, είτε παραλληλοποιώντας μία συλλογή αντικειμένων(π.χ. λίστα, πίνακα). Προσφέρονται δύο τύποι operations, τα transformations και τα actions. Τα transformations κατασκευάζουν ένα RDD από ένα άλλο RDD. Για παράδειγμα, ένα σύνηθες transformation είναι το filter, το οποίο φιλτράρει τα δεδομένα ενός RDD με βάση κάποιο

κριτήριο. Σε αντίθεση τα actions υπολογίζουν ένα αποτέλεσμα βασισμένο σε ένα RDD, το οποίο είτε το επιστρέφουν στο driver πρόγραμμα, είτε το αποθηκεύουν σε ένα εξωτερικό σύστημα αρχείων(π.χ. HDFS). Για παράδειγμα, ένα σύνηθες action είναι το first(), το οποίο επιστρέφει το πρώτο στοιχείο του RDD.

Τα transformations και τα actions είναι διαφορετικά, εξαιτίας του τρόπου με τον οποίο το Spark υπολογίζει τα RDDs. Αν και μπορούν να οριστούν νέα RDDs μέσω transformations οποιαδήποτε στιγμή, το Spark τα υπολογίζει την πρώτη φορά που θα χρησιμοποιηθούν σε ένα action(lazy evaluation). Αυτή η προσέγγιση μπορεί να φαίνεται παράξενη, αλλά έχει μεγάλο νόημα σε Big Data. Για παράδειγμα, αν χρειαζόταν να φιλτραριστεί ένα πολύ μεγάλο αρχείο κειμένου, ώστε να μείνουν μόνο οι σειρές που περιλαμβάνουν μία συγκεκριμένη λέξη(transformation/filter) και έπειτα να βρεθεί το πρώτο στοιχείο του RDD(action/first()), θα ήταν σπατάλη χρόνου και χώρου αποθήκευσης να γίνει η διαδικασία αυτή σειριακά. Στην πραγματικότητα, το Spark, σε αυτή την περίπτωση, θα σκάναρε το αρχικό αρχείο κειμένου έως ότου έβρισκε την πρώτη σειρά με τη λέξη, γλιτώνοντας με αυτό τον τρόπο πολύ χρόνο και χώρο αποθήκευσης.

Το lazy evaluation επιτυγχάνεται γιατί το Spark αποθηκεύει τις εξαρτήσεις μεταξύ των RDDs, κάθε φορά που γίνεται ένα transformation. Η καταγραφή των αλλεπάλληλων transformations γίνεται στο lineage graph[14](Εικόνα 13). Το Spark χρησιμοποιεί τις πληροφορίες που έχουν καταγραφεί σε περίπτωση που χρειάζεται να υπολογίσει ένα RDD(action) ή να ανακτήσει την τιμή μέρους ενός RDD.



Εικόνα 13 - RDD lineage graph [14]

Αν απαιτείται επαναχρησιμοποίηση ενός RDD πολλές φορές, δηλαδή εφαρμόζονται σε αυτό πολλά actions, το Spark δίνει τη δυνατότητα στο χρήστη να αποθηκεύει το RDD, με χρήση των εντολών RDD.persist() και RDD.cache(). Η αποθήκευση μπορεί να γίνει είτε στο δίσκο είτε στη μνήμη, όπου το RDD αποθηκεύεται διαχωρισμένο σε partitions στη μνήμη κάθε μηχανήματος του cluster.

Τέλος, ο χρήστης έχει τη δυνατότητα να φέρει μέρος ή και ολόκληρα RDDs από τους executors στο χώρο αποθήκευσης του driver, μέσω των actions take() και collect() αντίστοιχα. Η παραπάνω διαδικασία, όμως πρέπει να γίνεται με προσοχή, καθώς είναι απαραίτητο η μνήμη του driver να μπορεί να χωρέσει ολόκληρο το RDD, που προηγουμένως ήταν χωρισμένο στις μνήμες πολλών executors.

3.3.2. Dataset και DataFrame

Το Dataset είναι μια κατανεμημένη συλλογή δεδομένων, που προστέθηκε στο Spark 1.6. Παρέχει τα πλεονεκτήματα των RDDs σε συνδυασμό με τα πλεονεκτήματα της βελτιστοποιημένης μηχανής εκτέλεσης της Spark SQL. Ένα Dataset μπορεί να κατασκευαστεί από JVM objects και στη συνέχεια να μετασχηματιστεί μέσω transformations(π.χ. map, flatmap, filter). Η συγκεκριμένη μορφή δεδομένων είναι διαθέσιμη σε Scala και Java. Η Python δεν το υποστηρίζει, αλλά λόγω της δυναμικής της φύσης, πολλά από τα πλεονεκτήματα του είναι ήδη διαθέσιμα μέσω βιβλιοθηκών της(αντίστοιχα και στη γλώσσα R).

Το DataFrame είναι ένα σύνολο δεδομένων που οργανώνεται σε ονοματισμένες στήλες. Είναι εννοιολογικά ισοδύναμο με έναν πίνακα μιας βάσης δεδομένων και ουσιαστικά είναι ένα RDD αποτελούμενο από γραμμοπίνακες με πρόσθετες πληροφορίες σε κάθε στήλη(π.χ. τύπος ή όνομα στοιχείων). Τα DataFrames μπορούν να κατασκευαστούν είτε από δομημένα αρχεία δεδομένων, είτε από πίνακες του Hive, είτε από εξωτερικές βάσεις δεδομένων, είτε ακόμη και από υπάρχοντα RDDs. Είναι διαθέσιμα τόσο στις Scala και Java, όπου αναπαρίστανται από Datasets γραμμών, όσο και στις Python και R. Στη Scala, το DataFrame είναι ένα type alias του Dataset[Row]. Ενώ στη Java, οι χρήστες πρέπει να χρησιμοποιήσουν το Dataset <Row> για να αναπαραστήσουν ένα DataFrame[11].

3.4. Βιβλιοθήκες Apache Spark

3.4.1. Spark SQL

Η Spark SQL είναι η βιβλιοθήκη του Spark για τις εργασίες με δομημένα δεδομένα. Τα δομημένα δεδομένα είναι οποιαδήποτε δεδομένα που έχουν ένα σχήμα(schema), δηλαδή γνωστά πεδία για κάθε εγγραφή. Όταν υπάρχει αυτός ο τύπος δεδομένων, η Spark SQL κάνει πιο αποτελεσματική την φόρτωση και την αναζήτηση τους. Συγκεκριμένα, η Spark SQL παρέχει τρεις βασικές δυνατότητες: μπορεί να φορτώσει δεδομένα από μια ποικιλία δομημένων πηγών (π.χ. JSON, Hive, και Parquet), επιτρέπει την αναζήτηση δεδομένων χρησιμοποιώντας SQL και προσφέρει βέλτιστη σύνδεση μεταξύ SQL και Python/Java/Scala(Εικόνα 14), καθώς υποστηρίζεται με αυτό το τρόπο το join μεταξύ RDDs και SQL πίνακες. Για την επίτευξη των παραπάνω δυνατοτήτων, η Spark SQL διαχειρίζεται DataFrames(SchemaRDD), τα οποία αποθηκεύουν δεδομένα με πιο βέλτιστο τρόπο από τα RDDs και προσφέρουν νέα operations, όπως την ικανότητα να τρέχουν SQL ερωτήματα.

Εικόνα 14 - Συμβατότητα Spark SQL [16]

Η Spark SQL μπορεί να υποστηρίξει το Apache Hive, τη Hadoop SQL engine, η οποία επιτρέπει στο χρήστη να χρησιμοποιήσει πίνακες Hive, UDFs(user-defined functions), SerDes(serialization and deserialization formats) και τη γλώσσα ερωτημάτων του Hive(HiveQL). Όταν χρησιμοποιείται η Spark SQL υπάρχουν δύο επιλογές εκκίνησης ανάλογα με το αν είναι αναγκαία η υποστήριξη Hive. Συνίσταται η δημιουργία του HiveContext για να υπάρχει πρόσβαση στη HiveQL και στις λειτουργικότητες του Hive. Αντίθετα, σε περιπτώσεις που δημιουργούνται conflicts με τις εξαρτήσεις του Hive, ο χρήστης μπορεί να επιλέξει το βασικό SQLContext, το οποίο δε χρησιμοποιεί Hive. Ο πιο βέλτιστος τρόπος χρήσης της Spark SQL είναι μέσα σε μία Spark εφαρμογή. Αυτό επιτυγχάνεται όταν το HiveContext κατασκευάζεται βασισμένο στο SparkContext. Σε

επόμενο στάδιο, αν ο χρήστης θέλει να κάνει ένα ερώτημα σε ένα πίνακα, αρκεί να κάνει χρήση της μεθόδου sql() στο HiveContext ή στο SQLContext[14].

Η προσωρινή αποθήκευση στη Spark SQL(caching) λειτουργεί λίγο διαφορετικά από το συνηθισμένο, καθώς, δεδομένου ότι είναι γνωστοί οι τύποι κάθε στήλης, το Spark αποθηκεύει τα δεδομένα πιο βέλτιστα. Όταν αποθηκεύεται ένας πίνακας, η Spark SQL αναπαριστά τα δεδομένα του στη μνήμη χρησιμοποιώντας format στηλών. Επιπλέον, δίνεται η δυνατότητα αποθήκευσης πινάκων στη μνήμη μέσω δηλώσεων HiveQL.

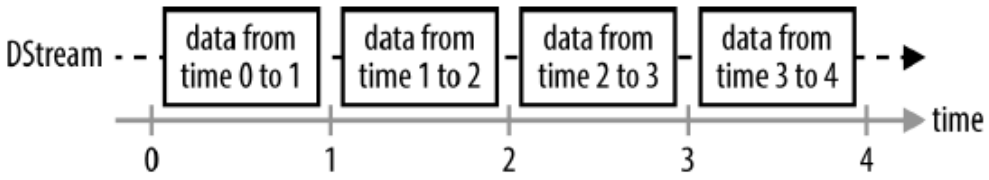
Σε πολλά pipelines, είναι εύκολο να συνδυαστεί η SQL (για τη συνοπτικότητά της) με κώδικα γραμμένο σε άλλες γλώσσες προγραμματισμού (για την ικανότητά τους να εκφράσουν πιο πολύπλοκη λογική). Όταν χρησιμοποιείται η Spark SQL για να επιτευχθεί ο παραπάνω συνδυασμός, προστίθενται και οι δυνατότητες διαχείρισης των DataFrames.

3.4.2. Spark Streaming

Πολλές εφαρμογές επωφελούνται όταν επεξεργάζονται τα δεδομένα την στιγμή που φθάνουν. Για μία εφαρμογή που παρακολουθεί στατιστικά δεδομένα σε πραγματικό χρόνο σχετικά με τις προβολές σελίδων, και πρέπει να εκπαιδεύσει ένα μοντέλο machine learning για παράδειγμα, είναι αναγκαία η βιβλιοθήκη Spark Streaming. Η συγκεκριμένη βιβλιοθήκη επιτρέπει στο χρήστη να γράφει streaming εφαρμογές χρησιμοποιώντας τα DStreams(discretized streams), ακολουθίες δεδομένων που φθάνουν σειριακά καθόλη τη διάρκεια της εφαρμογής. Κάθε DStream αναπαρίσταται από μία ακολουθία από RDDs που φθάνουν ανά συγκεκριμένο χρονικό βήμα και μπορεί να δημιουργηθεί από διάφορες πηγές εισόδου όπως Flume, Kafka και HDFS.

Προσφέρονται δύο τύποι operations, τα transformations, που δημιουργούν ένα καινούργιο DStream, και τα output operations, που γράφουν τα δεδομένα σε εξωτερικά συστήματα. Τα DStreams χρησιμοποιούν αρκετά από τα operations των RDDS, αλλά και νέα operations σχετικά με το χρόνο, όπως τα sliding windows.

Η Spark Streaming χρησιμοποιεί μια αρχιτεκτονική "micro-batch", όπου οι streaming υπολογισμοί αντιμετωπίζονται ως μία ακολουθία batch υπολογισμών σε μικρά σύνολα δεδομένων. Τα δεδομένα, που λαμβάνονται, οργανώνονται σε μικρότερα σύνολα δεδομένων με βάση μικρά χρονικά διαστήματα και δημιουργούν ισάριθμα RDDs(Εικόνα 15). Το μέγεθος αυτών των χρονικών διαστημάτων εξαρτάται από την παράμετρο batch interval, η οποία παίρνει τιμές από 500milliseconds έως και μερικά δευτερόλεπτα.



Εικόνα 15 - Είσοδος DStream [14]

Η εκκίνηση πραγματοποιείται δημιουργώντας το StreamingContext, το οποίο δέχεται ως παράμετρο εισόδου το batch interval. Στην συνέχεια, χρησιμοποιώντας τις κλήσεις start() και awaitTermination(), η εφαρμογή ξεκινάει τη λήψη δεδομένων και περιμένει την ολοκλήρωση του streaming υπολογισμού πριν τερματίσει.

Τα DStreams έχουν ανοχή σε σφάλματα όπως και τα RDDs, καθώς χρησιμοποιείται και σε αυτή την περίπτωση το lineage graph και επιπλέον από προεπιλογή, τα λαμβανόμενα δεδομένα αντιγράφονται σε δύο κόμβους, έτσι ώστε πιθανή αποτυχία ενός worker να μη δημιουργήσει πρόβλημα. Η Spark Streaming περιλαμβάνει επίσης έναν μηχανισμό που ονομάζεται checkpoint και αποθηκεύει περιοδικά την κατάσταση των δεδομένων σε ένα αξιόπιστο σύστημα αρχείων (π.χ. HDFS ή S3). Συνήθως, τα checkpoints ρυθμίζονται ανά 5 με 10 μικρά σύνολα δεδομένων και έτσι το Spark, σε περίπτωση χαμένων δεδομένων, αρκείται στο να επιστρέψει στο τελευταίο checkpoint[14].

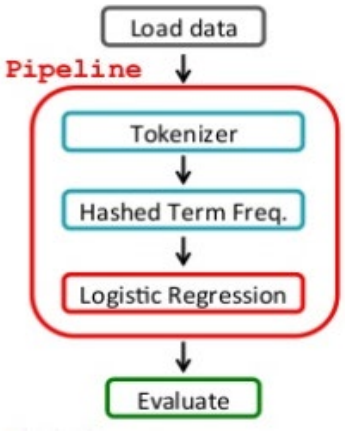
3.4.3. MLlib (Machine Learning library)

Η MLlib είναι η βιβλιοθήκη του Spark για συναρτήσεις μηχανικής μάθησης, η οποία έχει σχεδιαστεί για να τρέχει παράλληλα σε cluster χρησιμοποιώντας αλγορίθμους μάθησης και είναι διαθέσιμη σε όλες τις γλώσσες προγραμματισμού του Spark. Η φιλοσοφία της είναι απλή, καθώς προσφέρει τη δυνατότητα αλληλεπίδρασης μεταξύ διαφόρων αλγορίθμων και κατανεμημένων συνόλων δεδομένων, τα οποία αναπαρίστανται ως απλά RDDs ή DataFrames. Η MLlib αποτελείται ουσιαστικά από ένα αρκετά μεγάλο σύνολο συναρτήσεων που εφαρμόζονται πάνω σε RDDS και περιλαμβάνει μόνο παράλληλους αλγορίθμους όπως, για παράδειγμα, ο Distributed Random Forests, ο K-means και ο Alternating Least Squares. Είναι ιδανική σε περιπτώσεις που χρειάζεται ένα μεγάλο κατανεμημένο σύνολο δεδομένων για το training ενός μοντέλου.

Οι περισσότεροι αλγόριθμοι στην MLlib είναι επαναληπτικοί, χρησιμοποιώντας τα δεδομένα πολλές φορές. Έτσι, είναι πολύ σημαντικό να αποθηκεύονται(cache) τα RDDs εισόδου πριν περάσουν σε κάποιο αλγόριθμό της, ακόμη και αν χρειαστεί να αποθηκευτούν στο δίσκο λόγω μεγέθους. Στη Python, τα RDDS αποθηκεύονται αυτόματα,

ενώ στις Scala και Java η προσωρινή αποθήκευση των δεδομένων εξαρτάται από το χρήστη.

Αξίζει να σημειωθεί ότι υποστηρίζεται ένα υψηλότερου επιπέδου API(Application Program Interface) που ονομάζεται pipeline και περιλαμβάνει με τη σειρά τους αλγορίθμους(Εικόνα 16) που μετασχηματίζουν ένα σύνολο δεδομένων. Το API αυτό έχει την ικανότητα να αναζητάει αυτόματα το βέλτιστο συνδυασμό παραμέτρων που μπορεί να χρειάζονται κάποια στάδιά του ως είσοδο(π.χ. τον αριθμό των επαναλήψεων στο LogisticRegression). Το pipeline χρησιμοποιεί μία ομοιόμορφη αναπαράσταση των δεδομένων, καθώς διάφορα στάδια του μπορούν να προσθέτουν στήλες με επιπλέον πληροφορίες.



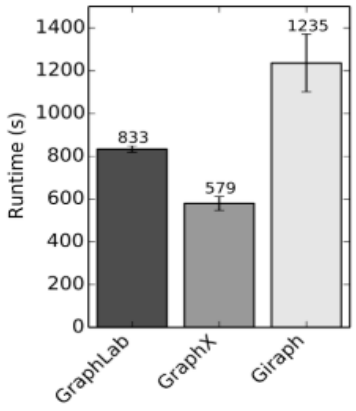
Εικόνα 16 - Pipeline API [17]

Αυτή την στιγμή στην MLlib περιλαμβάνονται δύο APIS, ένα που βασίζεται σε DataFrames και ένα παλαιότερο που βασίζεται σε RDDs, στα οποία περιλαμβάνονται αλγόριθμοι σχετικοί με στατιστική ανάλυση, εξαγωγή χαρακτηριστικών, classification, regression, clustering και άλλα.

3.4.4. GraphX

Η GraphX είναι η βιβλιοθήκη γραφημάτων του Spark και αναλαμβάνει graph-parallel και data-parallel υπολογισμούς. Προσφέρεται ευελιξία, καθώς ο χρήστης έχει τη δυνατότητα να εργαστεί τόσο με γραφήματα όσο και με συλλογές δεδομένων. Η GraphX συνδυάζει σε ένα ενιαίο σύστημα το ETL(Extract, Transform and Load) δεδομένων, τη λεπτομερή ανάλυση τους και τον επαναληπτικό υπολογισμό γραφημάτων. Επιπλέον, υποστηρίζονται η δημιουργία προσαρμοσμένων επαναληπτικών αλγορίθμων, χρησιμοποιώντας το API Pregel και τα transformations και joins μεταξύ γραφημάτων και RDDS. Το Spark προσφέρει

στη βιβλιοθήκη GraphX προσαρμοστικότητα, ανοχή σε σφάλματα και ευκολία χρήσης, μέσω των RDDs, κάνοντας εφικτή την σύγκριση του με άλλα βέλτιστα συστήματα γραφημάτων όπως το Apache Giraph και το Graphlab(Εικόνα 17).



Εικόνα 17 - PageRank performance [18]

Η GraphX χρησιμοποιεί in-memory υπολογισμούς, αξιοποιώντας τις βελτιστοποιήσεις που προσφέρουν τα συστήματα ροής δεδομένων(data flow systems), απλοποιεί τη διαδικασία κατασκευής γραφημάτων, καθώς προσφέρει τη δυνατότητα δυναμικού υπολογισμού των πολύ μεγάλων γραφημάτων. Τέλος αξίζει να σημειωθεί, ότι το σύνολο αλγορίθμων γραφημάτων που περιλαμβάνει μεγαλώνει, καθώς προστίθενται αλγόριθμοι ακόμη και από χρήστες, πολλαπλασιάζοντας τις δυνατότητες της GraphX.

3.5. Εκτέλεση Spark εφαρμογής

Αυτό το κεφάλαιο περιγράφει τον τρόπο διαμόρφωσης μιας εφαρμογής Spark, τα συστατικά εκτέλεσής της και τη μνήμη που καταλαμβάνει. Ακόμη αναλύεται ο τρόπος παρακολούθησής της εφαρμογής από το χρήστη, μέσω του Web UI.

3.5.1. Διαμόρφωση Spark

SparkConf

Το configuration του Spark συχνά πραγματοποιείται κατά το χρόνο εκτέλεσης της εφαρμογής. Ο κύριος μηχανισμός διαμόρφωσης στο Spark είναι η κλάση SparkConf, η οποία απαιτείται όταν δημιουργείται ένα νέο SparkContext. Η κλάση SparkConf είναι αρκετά απλή, καθώς περιέχει key/value ζεύγη από επιλογές διαμόρφωσης. Για να χρησιμοποιηθεί ένα αντικείμενο SparkConf, πρέπει αρχικά να προστεθούν τιμές διαμόρφωσης μέσω κλήσης της set() και έπειτα να προωθηθεί στον κατασκευαστή SparkContext. Για παράδειγμα, οι κλήσεις των setAppName() και setMaster() χρησιμοποιούνται για να οριστούν το spark.app.name και ο spark.master αντίστοιχα.

Spark-submit

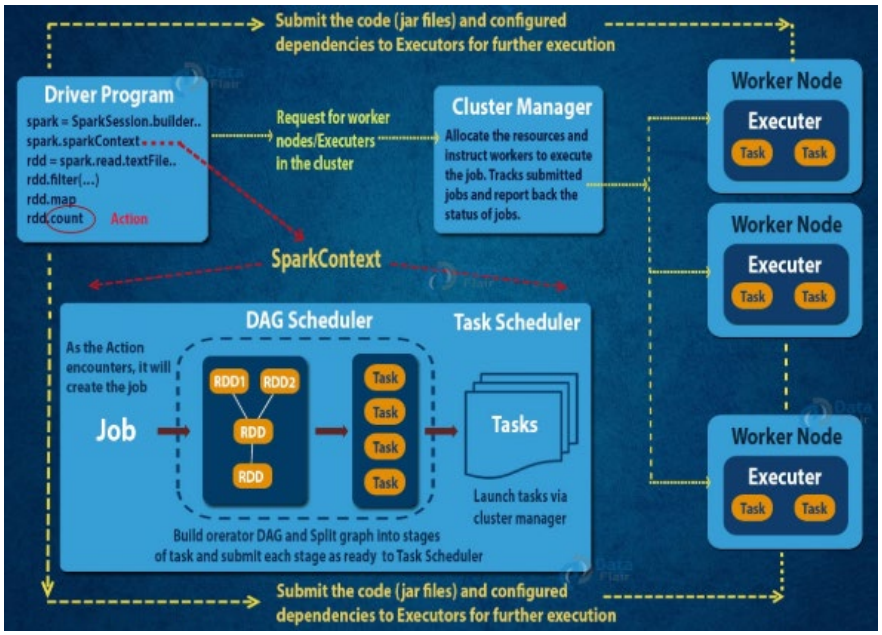
Στα προηγούμενα παραδείγματα, οι τιμές στο SparkConf ορίζονται προγραμματικά στον κώδικα, αλλά υπάρχει η δυνατότητα να συμπληρωθούν και δυναμικά την στιγμή εκκίνησης της εφαρμογής μέσω του spark-submit. Το εργαλείο spark-submit προσφέρει built-in παραμέτρους για τις πιο συνηθισμένες ρυθμίσεις του Spark, καθώς και μία πιο γενική παράμετρο --conf. Επιπλέον, υποστηρίζεται η φόρτωση των τιμών διαμόρφωσης από ένα αρχείο. Από προεπιλογή, λοιπόν, το spark-submit θα αναζητήσει πρώτα ρυθμίσεις γραμμένες στο αρχείο conf/spark-defaults.conf στο directory του Spark.

Το Spark έχει μια συγκεκριμένη σειρά προτεραιότητας στο τρόπο που επιλέγει ποιες ρυθμίσεις να εφαρμόσει κατά την εκτέλεση της εφαρμογής. Την υψηλότερη προτεραιότητα έχει το configuration μέσω του SparkConf αμέσως χαμηλότερη το configuration μέσω των παραμέτρων του spark-submit. Έπειτα, ακολουθεί το αρχείο conf/spark-defaults.conf και τέλος οι προεπιλεγμένες τιμές του Spark.

3.5.2. Συστατικά Εκτέλεσης - Jobs, Tasks και Stages

Μία εφαρμογή Spark κατά την εκτέλεσή της αναπαρίσταται τμηματικά σε πολλά jobs. Τα jobs είναι παράλληλοι υπολογισμοί που προκύπτουν σαν υλοποίηση ισάριθμων actions στον κώδικα. Ο κώδικας χρήστη ορίζει ένα κατευθυνόμενο ακυκλικό γράφημα(DAG), το οποίο αναπαριστά τα RDDs της εφαρμογής και τις εξαρτήσεις τους(operations). Καθώς καλείται ένα action, το υπάρχον DAG υποβάλλεται στο DAG Scheduler, ο οποίος στην συνέχεια χωρίζει το γράφημα σε πολλά stages. Στην απλούστερη περίπτωση, ο scheduler δημιουργεί ένα stage για κάθε job, αποτελούμενο από πολλά tasks. Όμως σε περιπτώσεις που υπάρχει repartition των δεδομένων(π.χ. ένα join δύο RDDs) ή αλλαγή στο locality των δεδομένων θα χρειαστεί το job να χωριστεί σε περισσότερα του ενός stage για να υλοποιηθεί. Σε υψηλότερο επίπεδο, υπάρχουν δύο τύποι RDD transformations, που επηρεάζουν τον αριθμό των stages ενός job. Τα narrow transformations(π.χ. map(), filter()) δεν κάνουν shuffle των δεδομένων ενός partition, οπότε ομαδοποιούνται σε ένα stage, σε αντίθεση με τα wide transformations(π.χ. reduceByKey()) που ομαδοποιούνται σε περισσότερα[19].

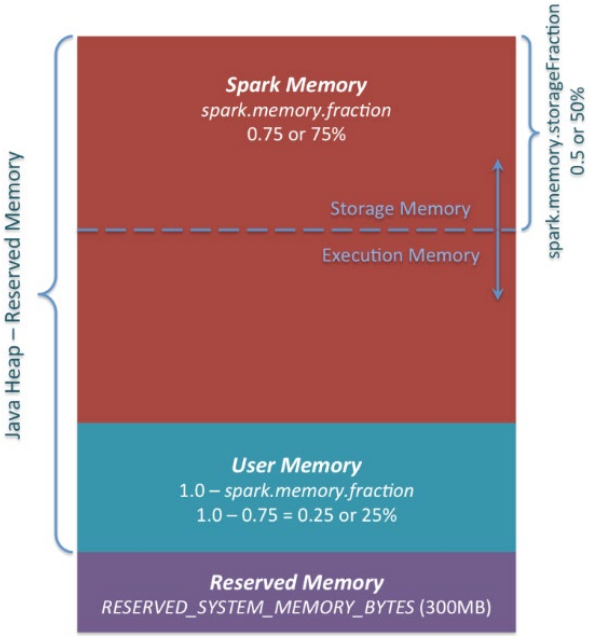
Τα stages προωθούνται στο Task Scheduler, ο οποίος εκκινεί tasks μέσω του Cluster Manager και έχει πλήρη άγνοια των εξαρτήσεων μεταξύ των stages. Κάθε stage χωρίζεται σε πολλά tasks, ίσα με τον αριθμό των partitions του RDD, τα οποία αναλαμβάνουν να εκτελέσουν τον ίδιο υπολογισμό αλλά στο δικό τους partition. Ο αριθμός των tasks που εκτελούνται ταυτόχρονα είναι ίσος με το πλήθος των cores όλων των executors(Εικόνα 18).



Εικόνα 18 - Παράδειγμα εκτέλεσης Spark εφαρμογής [19]

3.5.3. Διαχείριση Μνήμης

Η μνήμη που καταλαμβάνει μία εφαρμογή Spark απαρτίζεται από τα JVM heaps του driver και των executors. Το Spark χρησιμοποιεί τη μνήμη των executors με διάφορους τρόπους. Αρχικά, το σύστημα κρατάει 300mb του JVM heap, που ονομάζεται Reserved Memory. Ο υπόλοιπος χώρος, δίνεται στην εφαρμογή για την εκτέλεσή της. Το 75% του χώρου αυτού ονομάζεται Spark Memory και είναι η μνήμη που διαχειρίζεται το Spark, ενώ το υπόλοιπο 25% ονομάζεται User Memory και χρησιμοποιείται από τον κώδικα χρήστη(Εικόνα 19). Το Spark Memory χωρίζεται σε δύο κομμάτια. Το 50% ονομάζεται Storage Memory και είναι το κομμάτι στο οποίο αποθηκεύονται τα partitions των RDDs όταν χρησιμοποιούνται τα persist() και cache(). Το όριο αυτό ορίζεται από το spark.memory.storageFraction και αν εξαντληθεί τότε αυτόματα διαγράφονται παλαιότερα partitions από τη μνήμη. Το υπόλοιπο κομμάτι του ονομάζεται Execution Memory και χρησιμοποιείται για την αποθήκευση των shuffle buffers και hash tables, κατά την εκτέλεση των tasks στους executors[20].



Εικόνα 19 – Spark Memory Management [20]

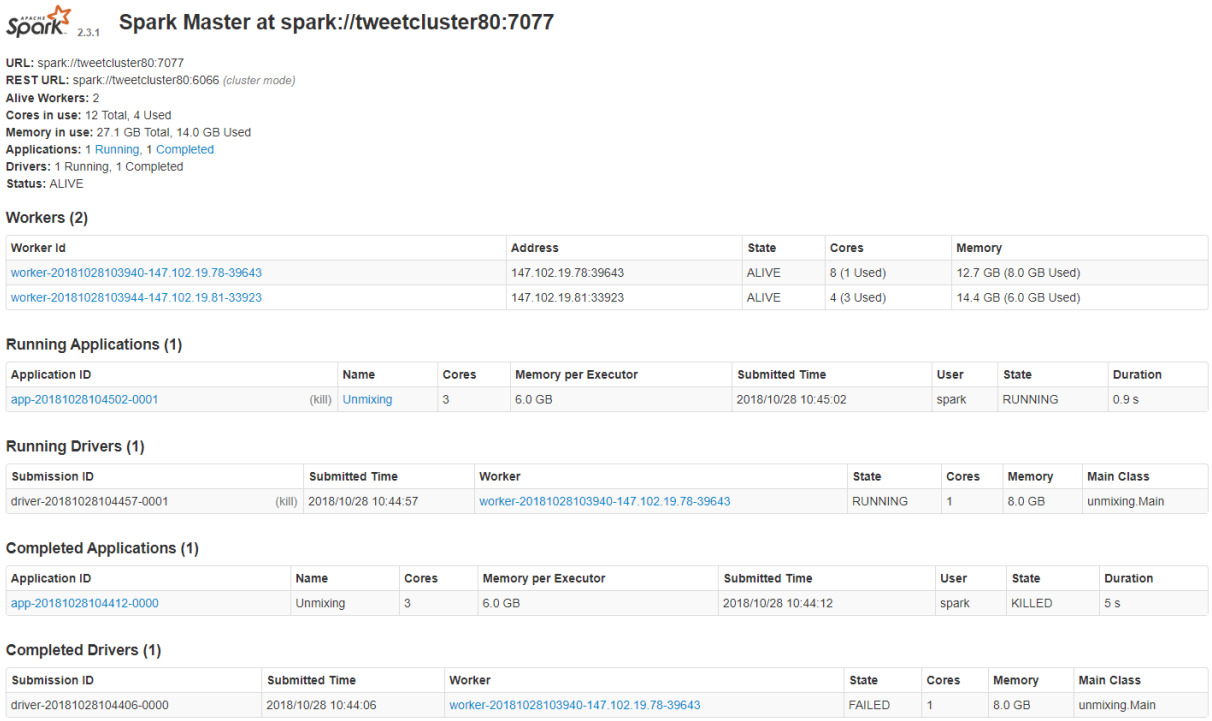
Το Spark προσφέρει τη δυνατότητα αποθήκευσης των RDDs σε πολλά επίπεδα αποθήκευσης. Όταν χρησιμοποιείται το cache(), τα RDDs αποθηκεύονται αυτόματα μόνο στη μνήμη(MEMORY_ONLY). Αντίθετα όταν χρησιμοποιείται το persist() ο χρήστης μπορεί επιλέξει το επίπεδο αποθήκευσης. Για παράδειγμα, το MEMORY_AND_DISK επίπεδο αποθήκευσης, σε περίπτωση που δεν αρκεί η μνήμη, στέλνει partitions στο δίσκο. Με αυτό το τρόπο τα partitions δε διαγράφονται, απλά επανέρχονται στη μνήμη μόνο σε

περίπτωση που ξαναχρησιμοποιηθούν. Αυτό είναι ιδιαίτερα χρήσιμο όταν ο επανυπολογισμός του RDD, που πιθανόν διαγράφηκε, είναι πολύ χρονοβόρος.

Τέλος, υποστηρίζονται και τα επίπεδα αποθήκευσης MEMORY_ONLY_SER και MEMORY_AND_DISK_SER, όπου τα δεδομένα αποθηκεύονται σειριοποιημένα. Η διαδικασία σειριοποίησης είναι λίγο πιο αργή, αλλά μπορεί να μειωθεί ο χρόνος που καταναλώνει ο garbage collector, καθώς πολλές μικρές εγγραφές αποθηκεύονται πλέον ως μία μεγαλύτερη σειριοποιήμενη. Αυτό συμβαίνει γιατί ο garbage collector καθυστερεί αισθητά καθώς αυξάνεται ο αριθμός των αντικειμένων στο JVM heap[14].

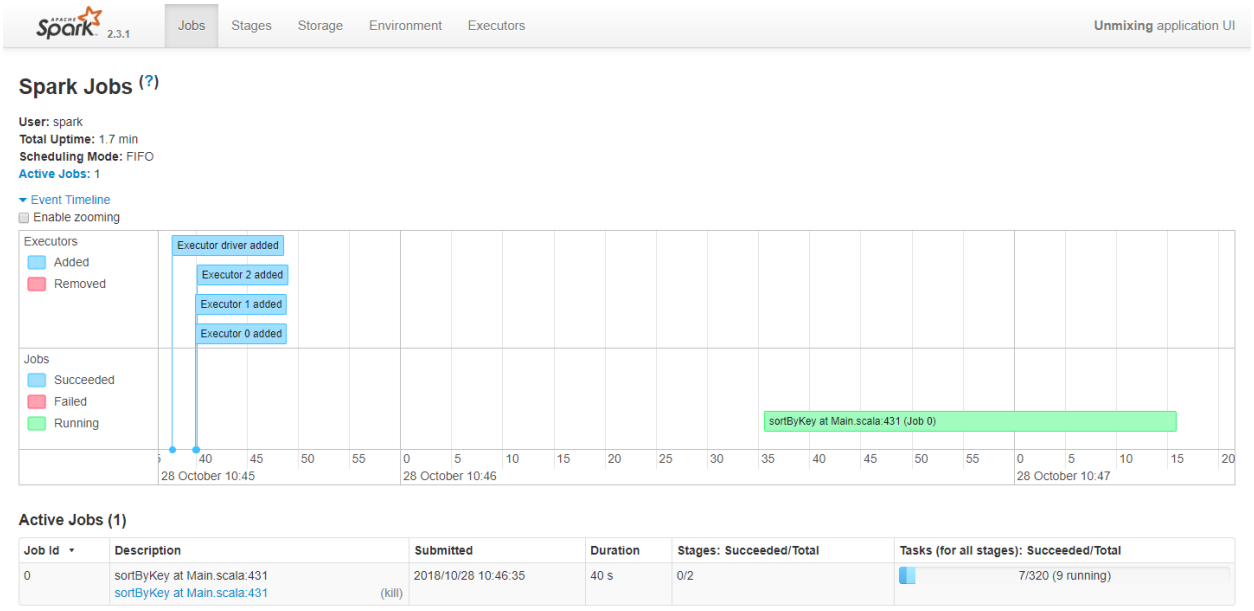
3.5.4. Monitoring - Spark Web UI

Ο βασικός τρόπος να γνωρίσει ο χρήστης το τρόπο που συμπεριφέρεται μία Spark εφαρμογή είναι να την παρακολουθήσει μέσω του Web UI που προσφέρει το Spark. Αρχικά, μέσω της πόρτας 8080(http://masternode:8080)(Εικόνα 20), που θεωρείται η σελίδα του master, δίνεται η δυνατότητα στο χρήστη να βλέπει την κατάσταση των workers(ALIVE/DEAD), τις IPs τους, τα cores και τη μνήμη που έχουν στη διάθεσή τους. Επιπλέον, ο χρήστης μπορεί να παρακολουθήσει το χρόνο εκτέλεσης των εφαρμογών τόσο αυτών που τρέχουν(Running Applications) όσο και αυτών που έχουν ήδη ολοκληρωθεί(Completed Applications).



Εικόνα 20 - Σελίδα του master

Επίσης, μέσω της πόρτας 4040(http://masternode:4040)(Εικόνα 21), που θεωρείται η σελίδα των jobs, μπορεί ο χρήστης να δει πληροφορίες σχετικά με την εκτέλεση των ενεργών και τον ολοκληρωμένων jobs του Spark. Μέσα στην σελίδα αυτή φαίνεται η πρόοδος των jobs, stages και tasks αναλυτικά, παρουσιάζοντας γραφήματα DAGs, χρόνους ολοκλήρωσης, αριθμό tasks και άλλα. Στόχος της είναι να γνωστοποιήσει στο χρήστη στατιστικά για την επίδοση της εφαρμογής, καθώς την αναλύει στα μικρότερα τμήματά της. Επιπλέον, υπάρχουν υποσελίδες για το Storage, το Environment και τους Executors όπου παρουσιάζονται πληροφορίες σχετικά με τα cached RDDs, τις παραμέτρους configuration του Spark περιβάλλοντος καθώς και τα μετρικά δεδομένα, τους πόρους και τον κώδικα που είναι προς εκτέλεση στους executors.



Εικόνα 21 - Σελίδα των jobs

ΚΕΦΑΛΑΙΟ 4 – ΥΛΟΠΟΙΗΣΗ SPECTRAL UNMIXING

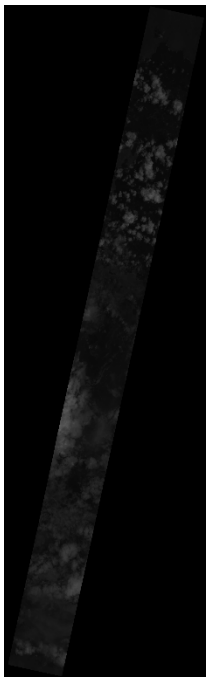
Στην παρούσα διπλωματική, χρησιμοποιήθηκε το εργαλείο Apache Spark για την υλοποίηση της διαδικασία του Hyperspectral Unmixing σε ένα cluster. Η υλοποίηση περιλαμβάνει έναν αλγόριθμο για εύρεση του πλήθους των endmembers της εικόνας εισόδου, έναν αλγόριθμο για της εξαγωγή των φασματικών υπογραφών τους και τέλος έναν αλγόριθμο για τον υπολογισμό των abundances που αντιστοιχούν σε αυτά.

4.1. Δεδομένα Εισόδου

Για το πειραματικό μέρος της διπλωματικής χρησιμοποιήθηκαν τόσο πραγματικά υπερφασματικά δεδομένα, όσο και συνθετικά δεδομένα.

Πραγματικά δεδομένα

Τα υπερφασματικά δεδομένα που χρησιμοποιήθηκαν παρέχονται από την ιστοσελίδα: https://earthexplorer.usgs.gov/ και συλλέχθηκαν από το υπερφασματικό εργαλείο Hyperion του δορυφόρου EO-1. Ο φασματικός κύβος που αναλύθηκε αποτελείται από 3.181x951=3.025.131 pixels, πραγματικού μεγέθους 30m το καθένα, και 242 κανάλια. Το συνολικό μέγεθος του κύβου είναι 1,5GB και αναπαρίσταται από 242 grayscale αρχεία(Εικόνα 22). Τα δεδομένα που Hyperion προσφέρονται σε GeoTIFF τύπο αρχείων, έχουν συλλεχθεί με φασματική σάρωση και αποτελούνται από 16-bit προσημασμένες ακέραιες τιμές ακτινοβολίας(radiance).



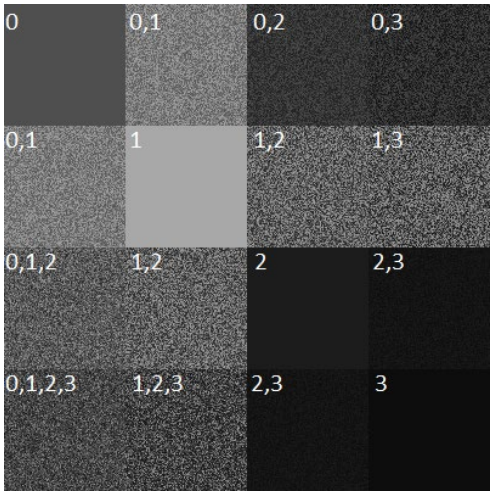
Εικόνα 22 - 13ο φασματικό κανάλι πραγματικής εικόνας

Συνθετικά Δεδομένα

Για την επαλήθευση και την επίδοση των αλγορίθμων κατασκευάστηκαν, μέσω της γλώσσας προγραμματισμού Python, συνθετικές υπερφασματικές εικόνες με χρήση φασματικών υπογραφών από τη βιβλιοθήκη[21] του USGS(United States Geological Survey) για το υπερφασματικό εργαλείο Hyperion. Οι φασματικές υπογραφές που προσφέρει η βιβλιοθήκη περιλαμβάνουν τιμές αντανακλάσεως υλικών(reflectance), σε 242 κανάλια, οι οποίες κατηγοριοποιούνται σε επτά ομάδες: Τεχνητά Υλικά, Επιχρίσματα, Υγρά, Ορυκτά, Οργανικές Ενώσεις, Εδάφη-Μίγματα και Βλάστηση. Επιλέχθηκαν φασματικές υπογραφές από διάφορες κατηγορίες ώστε να υπάρχει ρεαλιστική ποικιλομορφία στα συνθετικά δεδομένα.

Η συνθετική υπερφασματική εικόνα αποτελείται από 242 διαφορετικά κανάλια, που αναπαρίστανται με 242 grayscale tiff αρχεία. Το κάθε pixel της συνθετικής εικόνας αποτελεί γραμμικό συνδυασμό των τιμών των φασματικών υπογραφών για κάθε κανάλι. Πιο συγκεκριμένα, για κάθε pixel υπολογίζονται *n* ποσοστά τα οποία αντιστοιχούν στις *n* φασματικές υπογραφές από τις οποίες απαρτίζεται η εικόνα, και προκύπτουν από την κατανομή dirichlet έχοντας πάντα άθροισμα ίσο με 1. Η τιμή του pixel προκύπτει ως άθροισμα των γινομένων των *n* ποσοστών με την τιμή αντανακλάσεως της αντίστοιχης φασματικής υπογραφής.

Ο τρόπος με τον οποίο κατασκευάζεται η συνθετική εικόνα είναι τέτοιος ώστε να εμφανίζονται τόσο καθαρά(pure) pixels όσο και διάφοροι πιθανοί συνδυασμοί ανά δυάδες, τριάδες ή και παραπάνω από τα αρχικά υλικά που επιλέχθηκαν. Πιο συγκεκριμένα, η εικόνα χωρίζεται σε κομμάτια ίσα με το πλήθος των φασματικών υπογραφών υψωμένο στο τετράγωνο. Με αυτό τον τρόπο δημιουργείται ένας τετραγωνικός πίνακας *nxn*, όπου *n* το πλήθος των φασματικών υπογραφών, με κάθε θέση του να περιέχει ένα πλήθος από τα pixels της συνθετικής εικόνας.



Εικόνα 23 – Συνθετική εικόνα από 4 endmembers

Στη διαγώνιό του τοποθετούνται τα καθαρά pixels, δηλαδή pixels που περιέχουν 100% ποσοστό από ένα υλικό και 0% ποσοστό από τα υπόλοιπα ανά κανάλι. Δηλαδή για θέση (i,i) (όπου i=0,…,n-1) του πίνακα έχουμε pixels που αποτελούνται μόνο από το ιοστό υλικό. Επιπλέον ο άνω τριγωνικός πίνακας περιέχει pixels με συνδυασμό μόνο δύο υλικών και μας δίνει όλους τους πιθανούς συνδυασμούς ανά δυάδα, ενώ ο κάτω τριγωνικός περιέχει pixels αποτελούμενα από πιθανές δυάδες, τριάδες, τετράδες και ουτοκαθεξής. Για παράδειγμα, για θέση (i,j) (όπου i,j=0,…,n-1, με i<>j), στον άνω τριγωνικό εμφανίζεται τυχαίος συνδυασμός του i-στου και j-στου υλικού, ενώ στον κάτω τριγωνικό εμφανίζεται τυχαίος συνδυασμός από τα i έως και j υλικά. Τα παραπάνω γίνονται εμφανή στην Εικόνα 23, όπου φαίνονται οι συνδυασμοί τεσσάρων υλικών.

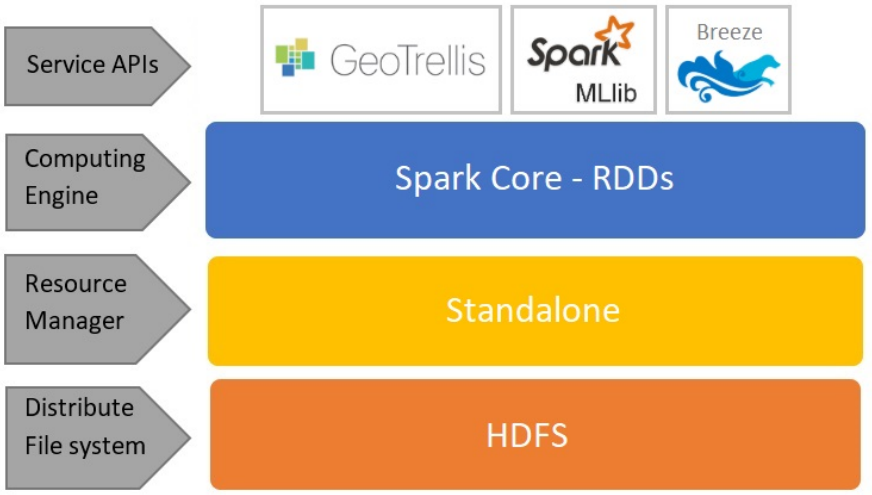
4.2. Πλατφόρμα Υλοποίησης

Για την υλοποίηση χρησιμοποιήθηκε ένα cluster, που αποτελείται από δυο μηχανήματα workers και ένα master. Αναλυτικά, τα χαρακτηριστικά των μηχανημάτων είναι:

	CPU	RAM
Master	2-Core Processor - Intel(R) Core(TM)2 Duo E7500 @2.93GHz	3GB
Worker 1	8-Core Processor - AMD FX(tm)-8120	13GB
Worker 2	4-core Processor - AMD Phenom(tm) II 965	15GB

Η γλώσσα προγραμματισμού, που χρησιμοποιήθηκε για τον κώδικα του Spark, είναι η Scala. Η Scala θεωρείται μια εξελιγμένη γλώσσα με ευέλικτη σύνταξη σε σύγκριση με την Java ή Python και είναι γρηγορότερη από την Python σε εφαρμογές επεξεργασίας και ανάλυσης δεδομένων λόγω του JVM.

Στην υλοποίηση της εφαρμογής χρησιμοποιήθηκε το κατανεμημένο σύστημα αρχείων του Hadoop(HDFS) και ο Standalone cluster manager, καθώς τα χαρακτηριστικά του εξυπηρέτησαν τις απαιτήσεις της εφαρμογής. Επιπλέον, χρησιμοποιήθηκαν οι βιβλιοθήκες MLlib του Spark, Breeze και Geotrellis. Από την MLlib έγινε χρήση αλγορίθμων και δομών δεδομένων, που έχουν ως βάση τα RDDs. Επιπλέον, η Breeze επέτρεψε την πραγματοποίηση σύνθετων μαθηματικών πράξεων μεταξύ πινάκων και στατιστικών τιμών. Τέλος, η βιβλιοθήκη Geotrellis βοήθησε στην εισαγωγή των GeoTIFF αρχείων στο Apache Spark. Στην Εικόνα 24 φαίνεται η συνολική πλατφόρμα που χρησιμοποιήθηκε.



Εικόνα 24 - Πλατφόρμα Υλοποίησης

Breeze

Οι υπολογιστικές πράξεις αυτής της εφαρμογής απαιτούν μία βιβλιοθήκη που να είναι ικανή να διαχειριστεί την πολυπλοκότητά τους. Η Breeze είναι μία από τις πιο γνωστές και ικανές βιβλιοθήκες γραμμικής άλγεβρας. Η MLlib, άλλωστε, έχει σχεδιαστεί με βάση τη Breeze, αλλά στην παρούσα διπλωματική έχουν χρησιμοποιηθεί και περαιτέρω ικανότητές της.

Geotrellis

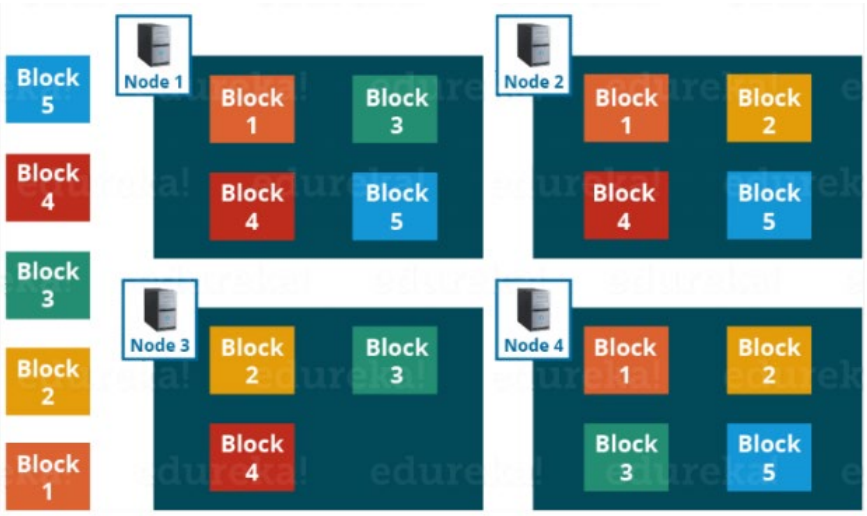
Η Geotrellis είναι μία βιβλιοθήκη της Scala σχεδιασμένη για ανάλυση γεωγραφικών δεδομένων με υψηλή επίδοση. Επιτρέπει την γρήγορη ανάγνωση, εγγραφή και επεξεργασία raster δεδομένων. Στην παρούσα διπλωματική χρησιμοποιήθηκε για την ανάγνωση των GeoTIFF αρχείων και τη μετατροπή τους σε RDDs, ώστε στην συνέχεια αυτά να επεξεργαστούν από την MLlib.

HDFS

Το Apache HDFS ή το Hadoop Distributed File System είναι ένα δομημένο σύστημα αρχείων, στο οποίο κάθε αρχείο χωρίζεται σε block ενός προκαθορισμένου μεγέθους. Αυτά τα blocks αποθηκεύονται σε ένα cluster από ένα ή περισσότερα μηχανήματα. Το HDFS ακολουθεί μια αρχιτεκτονική master/slave, όπου ένα cluster αποτελείται από ένα NameNode (master κόμβος) και πολλούς DataNodes (slave κόμβοι).

Ο NameNode είναι ο κύριος κόμβος της αρχιτεκτονικής Apache Hadoop HDFS που διατηρεί και διαχειρίζεται τα μπλοκ που υπάρχουν στους DataNodes. Επίσης, είναι ένας εξυπηρετητής που διαχειρίζεται το σύστημα αρχείων Namespace και ελέγχει την πρόσβαση στα αρχεία. Η αρχιτεκτονική HDFS είναι κατασκευασμένη με τέτοιο τρόπο ώστε τα δεδομένα χρήστη να μην παραμένουν ποτέ στο NameNode, αλλά να βρίσκονται μόνο στους DataNodes. Ο NameNode καταγράφει τα metadata όλων των αρχείων που είναι αποθηκευμένα στο cluster, για παράδειγμα τη θέση των αποθηκευμένων block, το μέγεθος των αρχείων, τα δικαιώματα και την ιεραρχία. Σε περίπτωση αποτυχίας ενός DataNode, ο NameNode επιλέγει νέους DataNodes για τα καινούργια αντίγραφα, ισορροπεί τη χρήση του δίσκου και διαχειρίζεται την επικοινωνία ανάμεσα στους DataNodes. Σε αντίθεση με το NameNode, ο DataNode είναι ένα μη δαπανηρό σύστημα που αποθηκεύει τα δεδομένα στο τοπικό σύστημα αρχείων(ext3 ή ext4). Ανά τακτά χρονικά διαστήματα στέλνει σήμα στο NameNode για να τον ενημερώσει σχετικά με την κατάσταση του HDFS.

Το HDFS παρέχει έναν αξιόπιστο τρόπο αποθήκευσης τεράστιων δεδομένων σε ένα κατανεμημένο περιβάλλον, καθώς δημιουργεί αντίγραφα των blocks για να υπάρχει ανοχή σε πιθανά σφάλματα. Ο προεπιλεγμένος αριθμός των αντιγράφων ορίζεται ως τρία, αλλά δίνεται η δυνατότητα στο χρήστη να τον αλλάξει(Εικόνα 25).



Εικόνα 25 - Αντίγραφα blocks - HDFS [22]

Τα παραπάνω χαρακτηριστικά του HDFS καθόρισαν τη χρησιμότητά του στην υλοποίηση της εφαρμογής.

4.3. Αλγόριθμος Προσδιορισμού Αριθμού Endmembers

4.3.1. Περιγραφή και Υλοποίηση Virtual Dimensionality

Περιγραφή

Ο αλγόριθμος που χρησιμοποιήθηκε στην παρούσα διπλωματική για τον προσδιορισμό του αριθμού των endmembers είναι ο Virtual Dimensionality. Επιλέχθηκε η μέθοδος HFC, καθώς χρησιμοποιείται ευρέως για τον προσδιορισμό των διαφορετικών φασματικών υπογραφών στην ανάλυση υπερφασματικής απεικόνισης[23].

Αρχικά, υποθέτοντας ότι ένα pixel του φασματικού κύβου έχει την μορφή $y = [r_1, \dots, r_L]$ όπου r_i είναι η ανακλαστικότητα του i-οστού καναλιού και L το πλήθος των καναλιών. Έτσι ο ορισμός μιας υπερφασματικής απεικόνισης δίνεται ως $Y = [y_1, \dots, y_N]$ όπου N είναι το πλήθος των pixels. Η μέθοδος αυτή βασίζεται στον υπολογισμό των ιδιοτιμών του correlation πίνακα(CM) και του covariance πίνακα(VM) των δεδομένων, που ορίζονται ως:

$$CM = \frac{(Y^T * Y)}{N-1}, \quad VM = \frac{(Y - \bar{Y})^T * (Y - \bar{Y})}{N}$$

Όπου το $\bar{Y}$ είναι το διάνυσμα των μέσων τιμών κάθε καναλιού.

Το επόμενο βήμα είναι ο υπολογισμός των ιδιοτιμών των δύο αυτών πινάκων. Κάθε πίνακας θα έχει L ιδιοτιμές, οπότε κάθε ιδιοτιμή λ_i συνδέεται με ένα συγκεκριμένο κανάλι. Η μέθοδος αυτή θεωρεί ότι η παρουσία μίας πηγής σήματος(endmember) στο i-οστό κανάλι εκφράζεται ως μία θετική διαφορά ανάμεσα στις ιδιοτιμές των CM και VM στο αντίστοιχο κανάλι.

$$\lambda_i^{CM} - \lambda_i^{VM} \geq 0$$

Ο αριθμός των endmembers υπολογίζεται μετρώντας, για κάθε κανάλι, πόσες φορές ισχύει η παραπάνω συνθήκη[24]. Όμως, η μέθοδος HFC ελέγχει για κάθε κανάλι πόσες φορές δεν ισχύει η παραπάνω συνθήκη δεδομένης μίας τιμής πιθανότητας P_{fa}(false alarm probability).

Στην Εικόνα 26 φαίνεται ο ψευδοκώδικας του αλγόριθμου HFC-VD, στον οποίο φαίνονται αναλυτικά τα βήματα τα οποία υλοποιήθηκαν. Στα βήματα 1-4 υπολογίζονται οι πίνακες CM και VM με τις ιδιοτιμές τους. Στη συνέχεια, για κάθε κανάλι υπολογίζεται η τυπική απόκλιση των ιδιοτιμών και ένα κατώφλι το οποίο δίνεται από την κανονική κατανομή με μέση τιμή 0, τυπική απόκλιση την υπολογισθείσα και πιθανότητα $P_{fa} = 10^{-5}$. Τέλος, το πλήθος των endmembers, ξεκινώντας μηδενικό, αυξάνεται κατά ένα στις περιπτώσεις όπου η διαφορά των ιδιοτιμών ξεπερνά το αντίστοιχο κατώφλι.

INPUT: $\mathbf{Y} = [\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N], P_{fa}$

1: $\mathrm{CM} = \frac{(\mathbf{Y}^T * \mathbf{Y})}{N-1}$;

2: $\mathrm{VM} = \frac{(\mathbf{Y}-\overline{\mathbf{Y}})^T * (\mathbf{Y}-\overline{\mathbf{Y}})}{N}$;

3: $\lambda^{\mathrm{CM}} = eig(\mathrm{CM})$; {compute the eigenvalues of CM}

4: $\lambda^{\mathrm{VM}} = eig(\mathrm{VM})$; {compute the eigenvalues of VM}

5: $dim = 0$; {initialize the number of endmembers}

6: **for** i:=1 **to** L **do**

7: $\sigma_i \cong \sqrt{\frac{2}{N}(\lambda_i^{\mathrm{CM}})^2 + \frac{2}{N}(\lambda_i^{\mathrm{VM}})^2}$;

8: solve $P_{fa} = \frac{1}{\sigma_i\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{z_i^2}{2\sigma_i^2}} dz_i$ to find x;

9: $diff = \lambda_i^{\mathrm{CM}} - \lambda_i^{\mathrm{VM}}$;

10: **if** $diff > x$ **then**

11: $dim = dim + 1$;

12: **end if**

13: **end for**

OUTPUT: dim

Εικόνα 26 - Αλγόριθμος HFC-VD [23]

Υλοποίηση

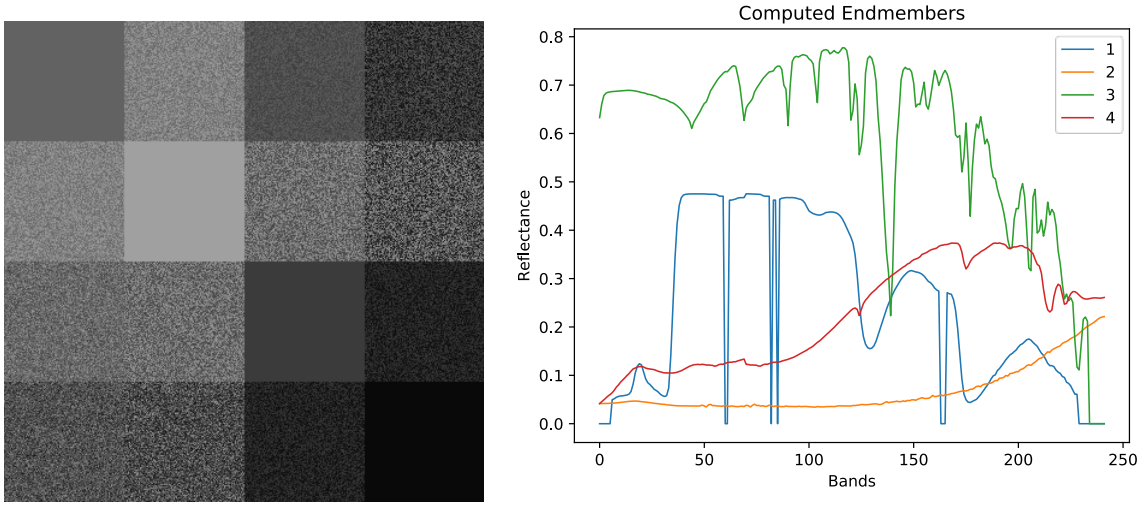
Στην προγραμματιστική υλοποίηση του VD, αρχικά τα GeoTIFF αρχεία εισάγονται ως ένα RDD με τη βοήθεια της βιβλιοθήκης Geotrellis. Εν συνεχεία, το RDD μετατρέπεται σε έναν κατανεμημένο πίνακα NxL της MLlib(RowMatrix), με σκοπό να γίνει ο υπολογισμός των CM και VM με τη χρήση της βιβλιοθήκης αυτής. Αξίζει εδώ να τονιστεί ότι οι πίνακες που προκύπτουν έχουν διαστάσεις LxL, γεγονός που καθιστά δυνατό τον υπολογισμό των ιδιοτιμών τους τοπικά με χρήση της βιβλιοθήκης Breeze. Τέλος, οι υπολογισμοί των διαφορών ανάμεσα στις ιδιοτιμές και της τυπικής απόκλισης πραγματοποιούνται μέσω RDDs, ώστε να υπάρχει όσο το δυνατόν καλύτερη επίδοση.

4.3.2. Επαλήθευση Virtual Dimensionality

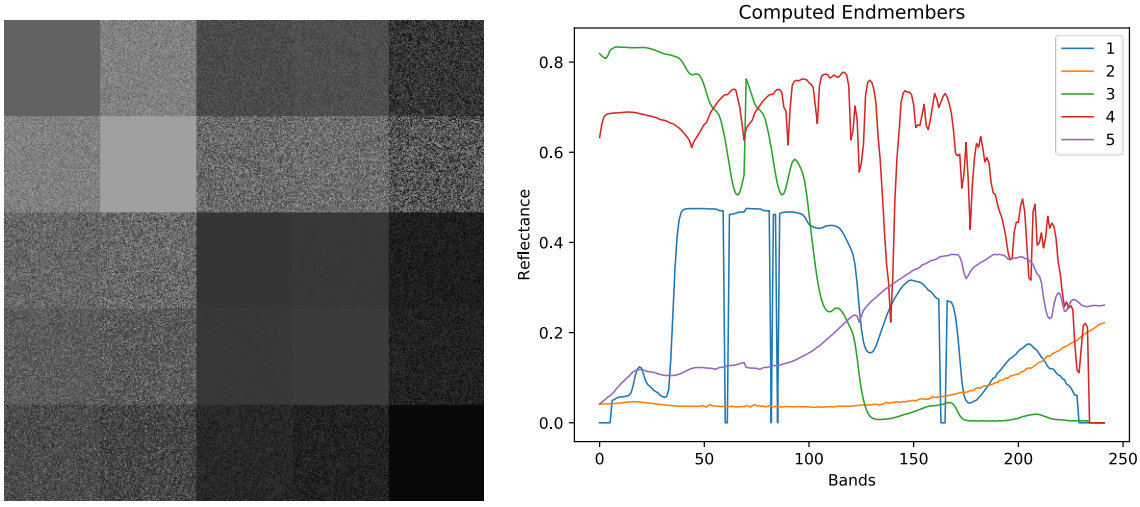
Στη διαδικασία επαλήθευσης του HFC-VD πραγματοποιήθηκαν πειράματα με διάφορες συνθετικές εικόνες. Δημιουργήθηκαν εικόνες τεσσάρων έως και οχτώ υλικών, τα οποία

επιλέχθηκαν από διαφορετικές κατηγορίες της βιβλιοθήκης του USGS, ώστε να προσομοιάζουν ρεαλιστικές απεικονίσεις. Τα υλικά αυτά είναι: Acetylene, Acmite, Actinolite, Aspen, Black Mn Coat, Melting Snow, Polyester και Rangeland.

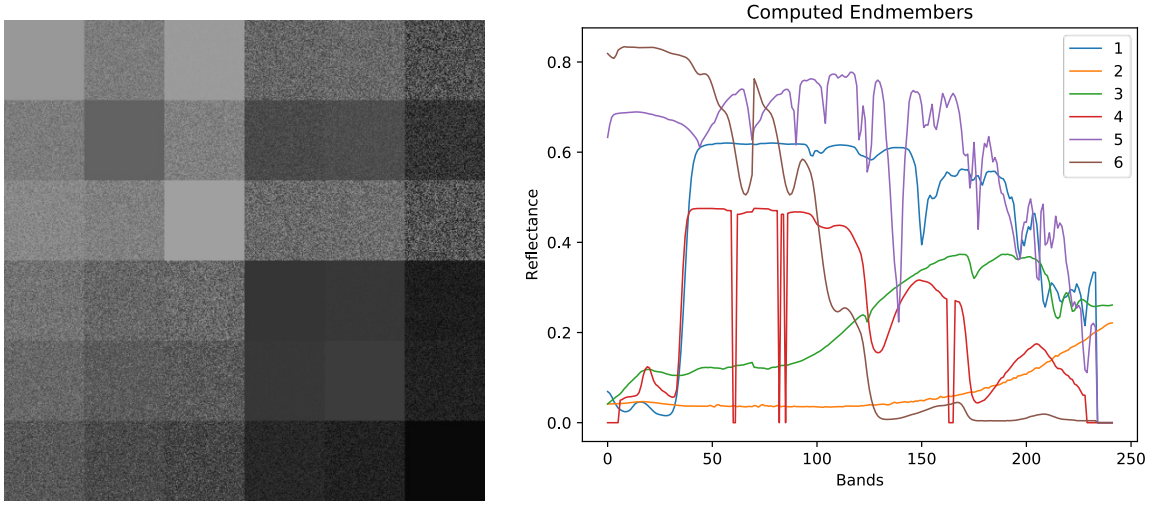
Παρακάτω παρουσιάζονται τα αποτελέσματα πέντε πειραμάτων για εικόνες με τέσσερα, πέντε, έξι, εφτά και οχτώ υλικά με μέγεθος 160.000, 250.000, 360.000, 490.000 και 640.000 pixels αντίστοιχα.



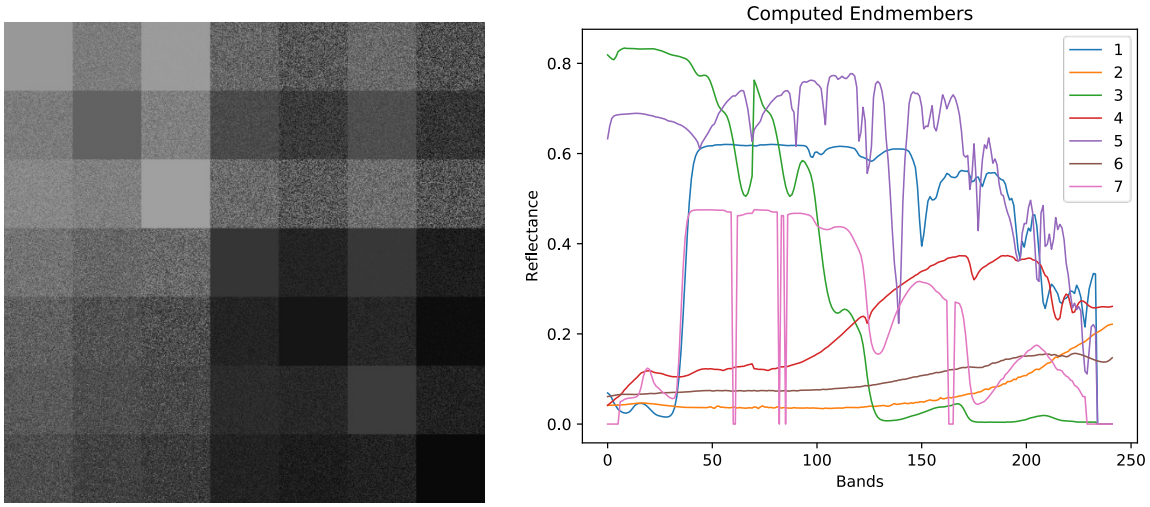
Εικόνα 27 – Πείραμα: 4 endmembers



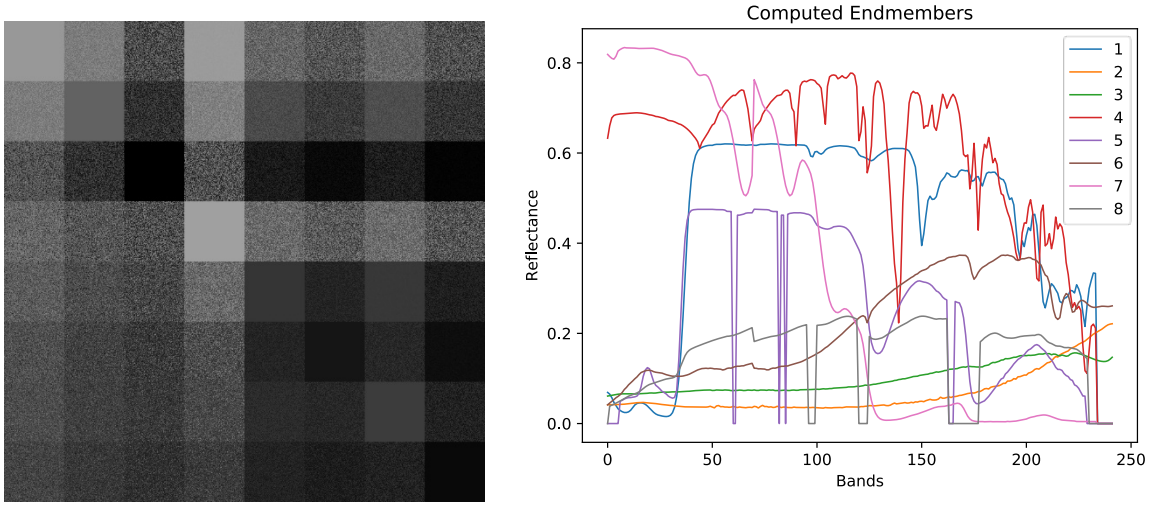
Εικόνα 28 – Πείραμα: 5 endmembers



Εικόνα 29 – Πείραμα: 6 endmembers



Εικόνα 30 – Πείραμα: 7 endmembers



Εικόνα 31 – Πείραμα: 8 endmembers

Με βάση τα αποτελέσματα των πέντε πειραμάτων είναι φανερό ότι η εφαρμογή ανταποκρίνεται βέλτιστα στην εύρεση του αριθμού των endmembers, όταν δέχεται ως είσοδο συνθετικά δεδομένα. Παρακάτω πραγματοποιείται μία δοκιμή σε πραγματικά δεδομένα.

4.4. Αλγόριθμος Εξαγωγής Endmember

4.4.1. Περιγραφή και Υλοποίηση Αλγορίθμου

Περιγραφή

Στην παρούσα εργασία χρησιμοποιήθηκε μία μέθοδος εξαγωγής endmember η οποία εκμεταλλεύεται την ικανότητα μείωσης των διαστάσεων του PCA και προτείνεται από το paper[25]. Ο αλγόριθμος PCA(Principal Component Analysis), αποκαλούμενος επίσης ως μετασχηματισμός Karhunen-Loeve, είναι μία ισχυρή τεχνική επεξεργασίας πολυδιάστατων δεδομένων με στόχο την αυτόματη μείωση των διαστάσεών τους. Έχει εφαρμοστεί σε μία μεγάλη ποικιλία διαφορετικών εφαρμογών όπως η απεικόνιση δεδομένων μεγάλων διαστάσεων, η λογιστική παλινδρόμηση (regression) και η εξαγωγή χαρακτηριστικών (feature extraction) στην αναγνώριση προτύπων. Ο αλγόριθμος που υλοποιήθηκε, λοιπόν, αποτελεί μία simplex-based μέθοδο, η οποία αξιολογεί τα principal components για την εξαγωγή των endmembers.

Στη βιβλιογραφία αναφέρεται ότι το πρώτο endmember το οποίο εξάγεται από τον SGA(Simplex Growing Algorithm) είναι πάντα ένα pixel που έχει είτε μέγιστη είτε ελάχιστη τιμή στην πρώτη συνιστώσα των μειωμένων διαστάσεων[26]. Με βάση αυτή την παραδοχή, η μέθοδος, που υλοποιήθηκε, χρησιμοποιεί τα πρώτα $p-1$ principal components του PCA, με p τον αριθμό των γραμμικά ανεξάρτητων endmembers, για να ορίσει ένα χώρο πολύ μικρότερης διάστασης από αυτόν του αρχικού υπερφασματικού χώρου. Δεδομένου ότι τα endmembers βρίσκονται στις κορυφές του simplex που ορίζεται από τα δεδομένα με διαστάσεις $p-1$, ένα υποσύνολο των min και max τιμών των πρώτων $p-1$ principal components αντιστοιχεί στις κορυφές αυτού του simplex. Για την εύρεση του βέλτιστου υποσυνόλου, η μέθοδος χρησιμοποιεί τεχνικές κατηγοριοποίησης και μεγιστοποίησης όγκου.

Αρχικά, έχει υπολογιστεί από τον αλγόριθμο HFC-VD το πλήθος p των endmembers στην εικόνα. Έχοντας αυτό τον αριθμό, μειώνονται οι διαστάσεις των δεδομένων από L σε $p-1$, με χρήση του αλγορίθμου PCA. Στην συνέχεια, συλλέγονται $2p-2$ pixels, τα οποία αντιστοιχούν στα pixels με τις ελάχιστες και μέγιστες τιμές των $p-1$ διαστάσεων. Τα pixels αυτά αποτελούν τα υποψήφια καθαρά(pure) pixels, δηλαδή τα endmembers, καθώς βρίσκονται στις κορυφές του φασματικού υποχώρου των $p-1$ διαστάσεων. Σε επόμενο βήμα υπολογίζονται οι SAD μεταξύ όλων των υποψήφιων endmembers και επιλέγονται όσες είναι παραπάνω από μία μοίρα. Τέλος, αν το πλήθος των επιλεγμένων endmembers είναι μεγαλύτερο από p, επιλέγονται τα p που δημιουργούν τον μέγιστο όγκο στο χώρο.

Υλοποίηση

Ο αλγόριθμος εξαγωγής endmember που υλοποιήθηκε ξεκινάει με τη μείωση των διαστάσεων των δεδομένων σε $p-1$, με χρήση του αλγορίθμου PCA. Επιλέχθηκε ο υλοποιημένος αλγόριθμος PCA της βιβλιοθήκης MLlib του Spark, ο οποίος χρησιμοποιεί σαν βασική δομή δεδομένων τα RDDs, δίνοντας σαν αποτέλεσμα έναν κατανεμημένο πίνακα RowMatrix, ο οποίος αναπαριστά τα δεδομένα σε έναν υποχώρο $p-1$ διαστάσεων. Στην συνέχεια χρησιμοποιώντας συναρτήσεις της δομής δεδομένων RDD συλλέγονται τα pixels με τις min και max τιμές όλων των principal components και μετατρέπονται ξανά σε έναν κατανεμημένο πίνακα RowMatrix, έτσι ώστε να υπολογιστούν όλες οι SAD μεταξύ τους, με χρήση της βιβλιοθήκης γραμμικής άλγεβρας της MLlib. Τέλος, επιλέγονται τα endmembers τα οποία μεγιστοποιούν τον όγκο στο χώρο $p-1$. Η διαδικασία αυτή πραγματοποιείται, αναγνωρίζοντας ως διαφορετικά pixels εκείνα τα οποία έχουν SAD μεγαλύτερη της μίας μοίρας και διαγράφοντας όσα θεωρήθηκαν διπλότυπα. Τα pixels που μένουν αποτελούν τα endmembers της εικόνας.

4.4.2. Επαλήθευση Αλγορίθμου

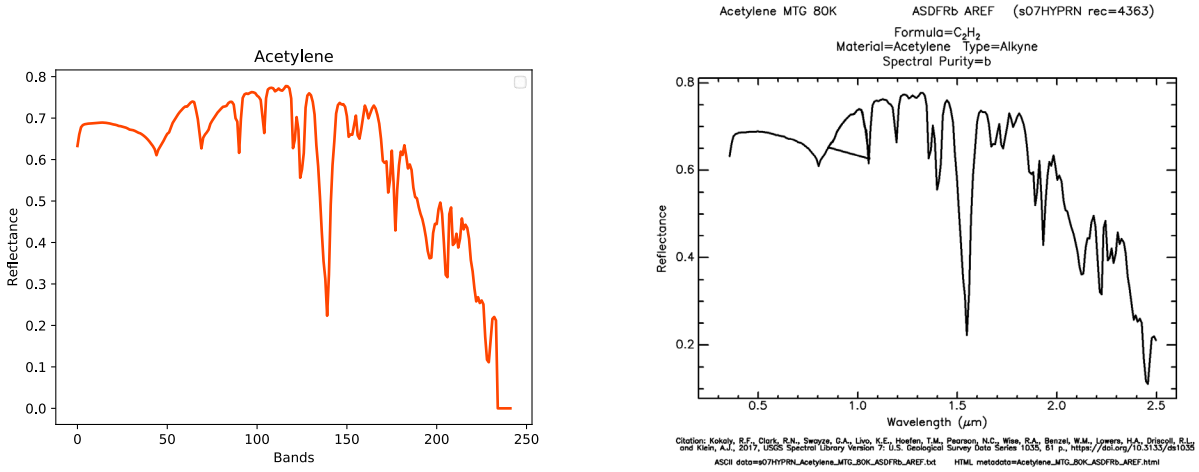
Για την επαλήθευση της εξαγωγής των ορθών endmembers, πραγματοποιήθηκαν πειράματα, τα οποία έδωσαν σαν αποτέλεσμα τις φασματικές υπογραφές των endmembers που προέκυψαν από τον αλγόριθμο. Τα αποτελέσματα των παραπάνω πειραμάτων συγκρίθηκαν με τις φασματικές υπογραφές των πραγματικών υλικών, που δόθηκαν από τη βιβλιοθήκη του USGS. Η φασματική υπογραφή ενός υλικού είναι το διάγραμμα της ανακλαστικότητάς του ως προς το μήκος κύματος ή τα φασματικά κανάλια.

Παρακάτω παρουσιάζονται συγκριτικά αποτελέσματα δύο πειραμάτων με συνθετικά δεδομένα. Το πρώτο πείραμα έγινε με την εικόνα των τεσσάρων υλικών (160.000 pixels) και το δεύτερο με την εικόνα των οχτώ υλικών (640.000 pixels).

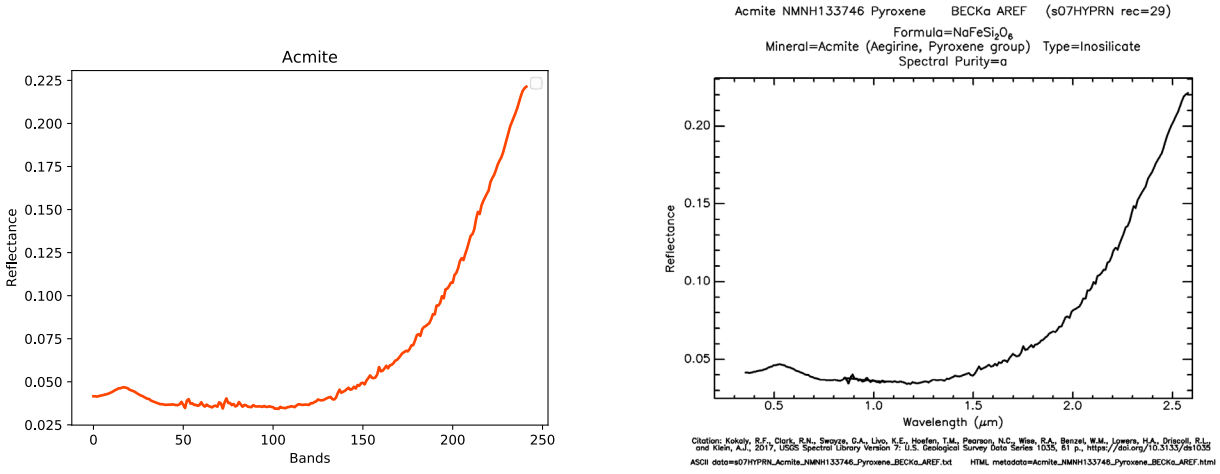
Στο πρώτο πείραμα χρησιμοποιήθηκαν τα υλικά:

Υλικά	*Κατηγορία Υλικού*
Acetylene	Organic Compounds
Acmite	Minerals
Actinolite	Soils & Mixtures
Aspen	Vegetation

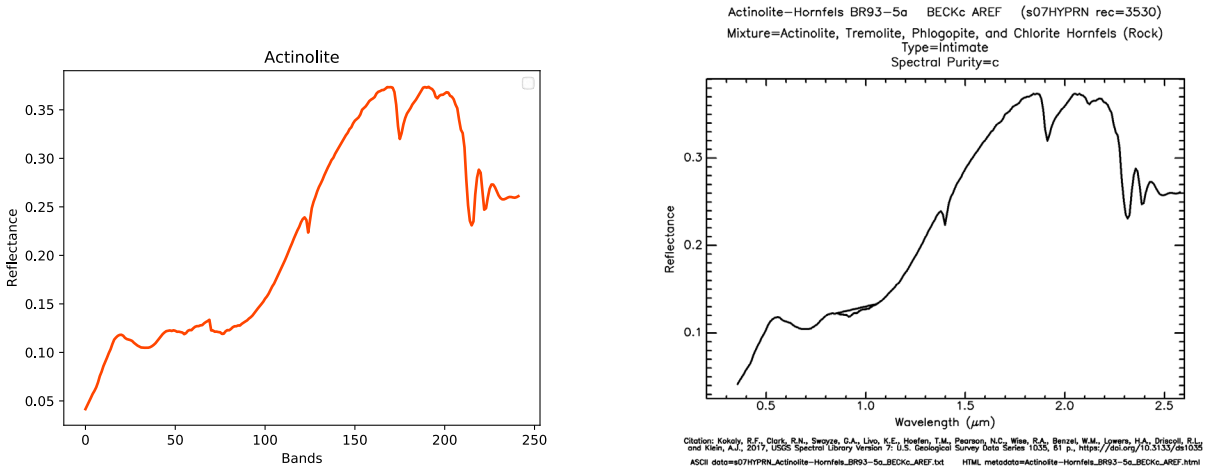
Παρακάτω δίνονται οι τέσσερις φασματικές υπογραφές που αντιστοιχούν στα τέσσερα υλικά της συνθετικής εικόνας. Αριστερά εμφανίζεται η φασματική υπογραφή, η οποία προέκυψε από την υλοποίηση, ενώ δεξιά εμφανίζεται η φασματική υπογραφή του υλικού, όπως αυτή δόθηκε από τη βιβλιοθήκη του USGS. Όπως φαίνεται από τις γραφικές τα endmembers που υπολογίστηκαν είναι πανομοιότυπα με τα αυθεντικά. Αξίζει εδώ να σημειωθεί ότι οι μηδενικές τιμές ανακλαστικότητας(reflectance) που εμφανίζονται στις γραφικές, αντιστοιχούν στα κακά κανάλια(bad bands). Τα bad bands προκύπτουν από λάθος μέτρηση του υπερφασματικού αισθητήρα και προτείνεται να αγνοούνται.



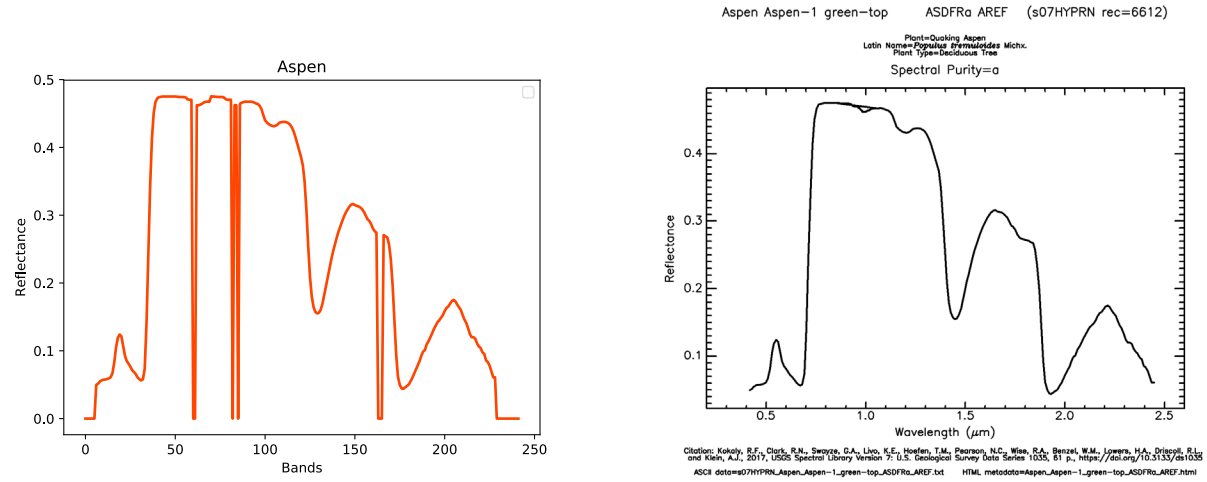
Εικόνα 32 - Πείραμα 1: Φασματική Υπογραφή Acetylene



Εικόνα 33 - Πείραμα 1: Φασματική Υπογραφή Acmite



Εικόνα 34 - Πείραμα 1: Φασματική Υπογραφή Actinolite

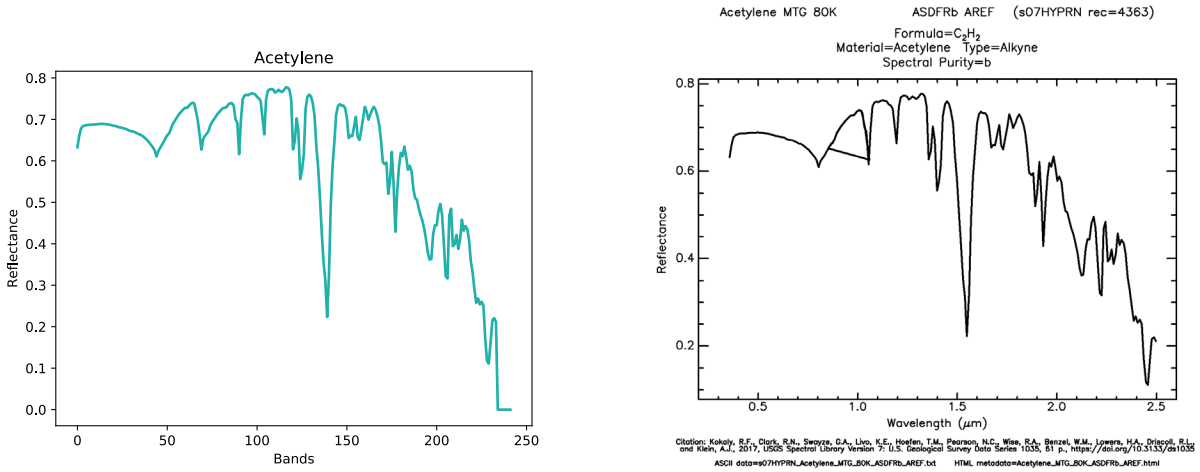


Εικόνα 35 - Πείραμα 1: Φασματική Υπογραφή Aspen

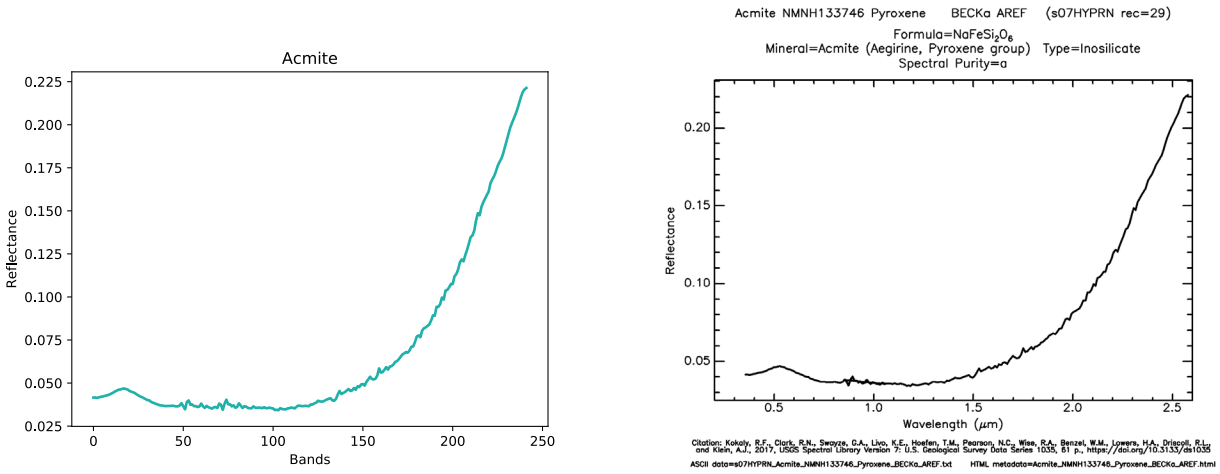
Στο δεύτερο πείραμα χρησιμοποιήθηκαν τα υλικά:

Υλικά	*Κατηγορία Υλικού*
Acetylene	Organic Compounds
Acmite	Minerals
Actinolite	Soils & Mixtures
Aspen	Vegetation
Blk Mn Coat	Coatings
Melting Snow	Liquids
Polyester	Artificial
Rangeland	Vegetation

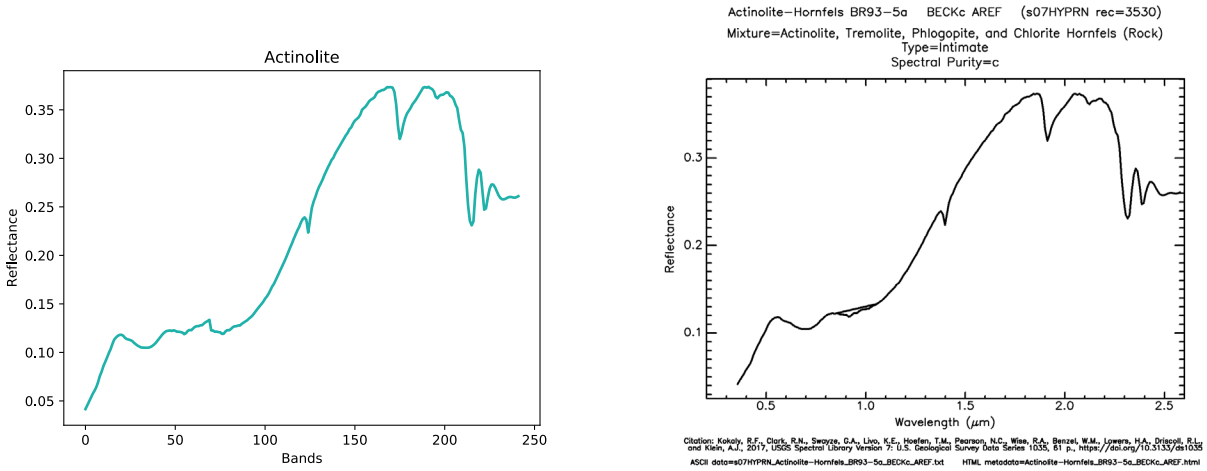
Παρακάτω δίνονται οι φασματικές υπογραφές που αντιστοιχούν στα οχτώ παραπάνω υλικά. Αριστερά εμφανίζεται η φασματική υπογραφή που προέκυψε από την υλοποίηση, ενώ δεξιά εμφανίζεται η φασματική υπογραφή του υλικού από τη βιβλιοθήκη του USGS. Και εδώ, παρόλο που η εικόνα περιείχε περισσότερα endmembers, ο αλγόριθμος έδωσε πανομοιότυπες τιμές. Οι μηδενικές τιμές του reflectance αντιστοιχούν και πάλι στα bad bands.



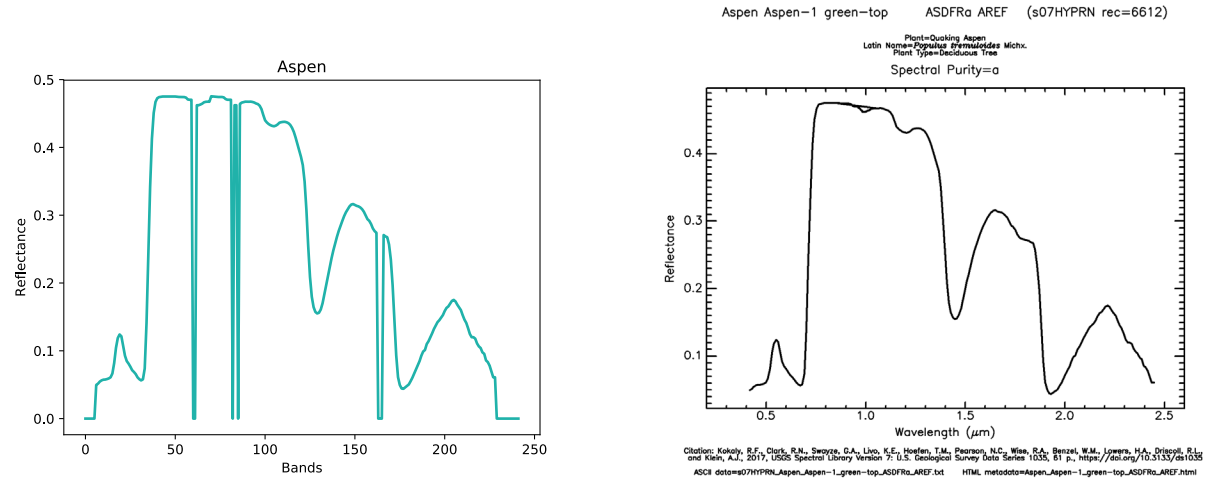
Εικόνα 36 - Πείραμα 2: Φασματική Υπογραφή Acetylene



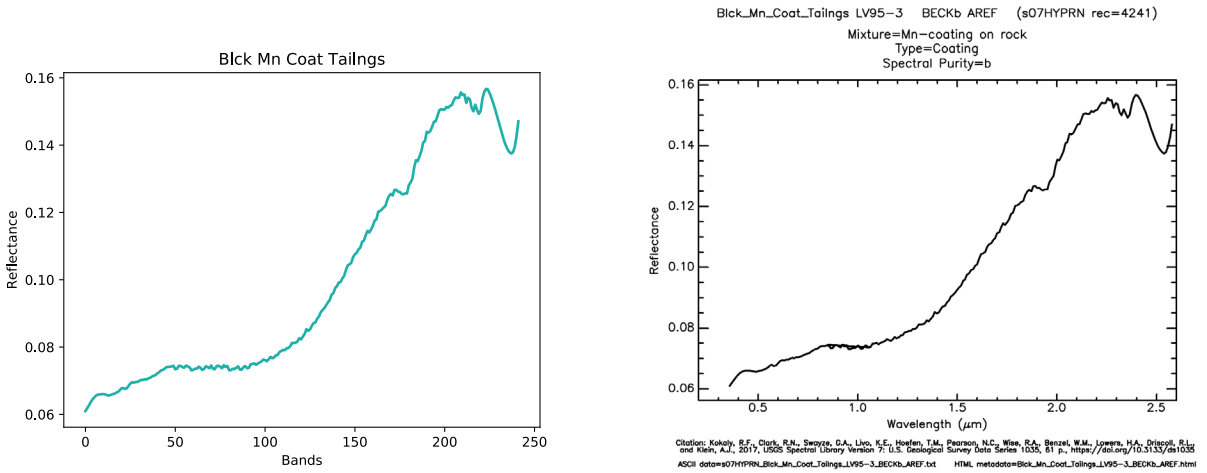
Εικόνα 37 - Πείραμα 2: Φασματική Υπογραφή Acmite



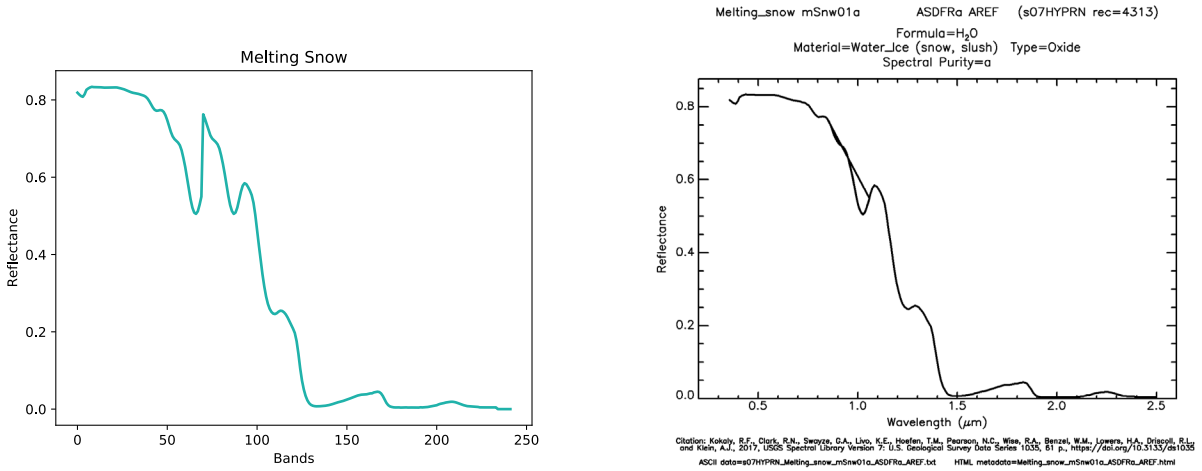
Εικόνα 38 - Πείραμα 2: Φασματική Υπογραφή Actinolite



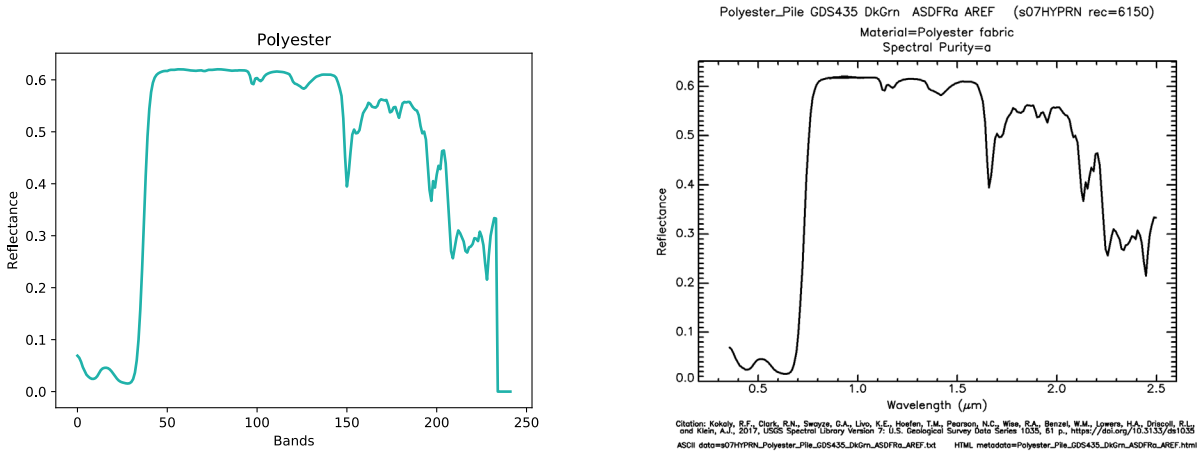
Εικόνα 39 - Πείραμα 2: Φασματική Υπογραφή Aspen



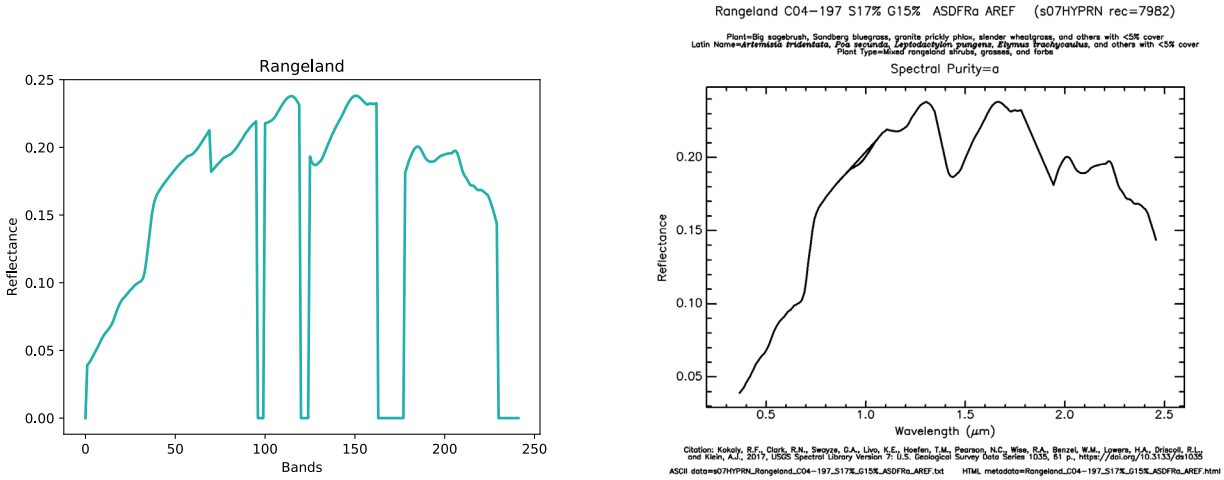
Εικόνα 40 - Πείραμα 2: Φασματική Υπογραφή Blck Mn Coat



Εικόνα 41 - Πείραμα 2: Φασματική Υπογραφή Melting Snow



Εικόνα 42 - Πείραμα 2: Φασματική Υπογραφή Polyester



Εικόνα 43 - Πείραμα 2: Φασματική Υπογραφή Rangeland

4.5. Αλγόριθμος Προσδιορισμού Abundances

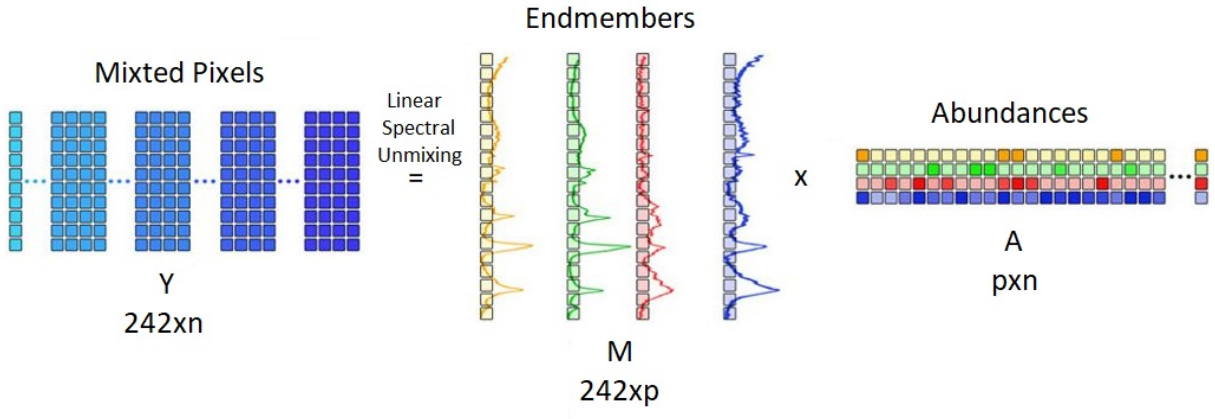
4.5.1. Περιγραφή και Υλοποίηση Μεθόδου Least Squares

Περιγραφή

Στην παρούσα διπλωματική εργασία για τον υπολογισμό των abundances χρησιμοποιήθηκε η μέθοδος Linear Least Squares, χωρίς τη χρήση περιορισμών (unconstrained least squares-UCLS). Το πρόβλημα Linear Spectral Unmixing(LSU) εκφράζεται ως:

$$Y = MA + N$$

Όπου Y είναι ο πίνακας που απεικονίζει τον φασματικό κύβο(διαστάσεων $242xn, n$ πλήθος pixels), M ο πίνακας με τα endmembers(διαστάσεων $242xp, p$ πλήθος endmembers), A ο πίνακας των abundances(διαστάσεων pxn), που χρειάζεται να υπολογιστεί, και N ο θόρυβος-σφάλμα(Εικόνα 44).



Εικόνα 44 - Linear Spectral Unmixing

Η μέθοδος Linear Least Squares υπολογίζει τις αφθονίες ελαχιστοποιώντας το σφάλμα ανακατασκευής:

$$e = \|Y - MA\|^2$$

Το παραπάνω πρόβλημα προκύπτει από την ανάλυση στατιστικής παλινδρόμησης (statistical regression analysis) και έχει λύση κλειστής μορφής[27], που είναι η:

$$A = (M^T M)^{-1} M^T Y \quad (1)$$

Υλοποίηση

Η παραπάνω μέθοδος υλοποιήθηκε με τη χρήση των βιβλιοθηκών Breeze και MLlib καθώς περιλαμβάνει άλγεβρα πινάκων με πολύ μεγάλο μέγεθος. Η έκφραση (1) περιέχει πολλαπλασιασμούς πολύ μεγάλων πινάκων, για αυτό το λόγο μετασχηματίστηκε, με χρήση της ιδιότητας $(AB)^T = B^T A^T$, σε αυτή:

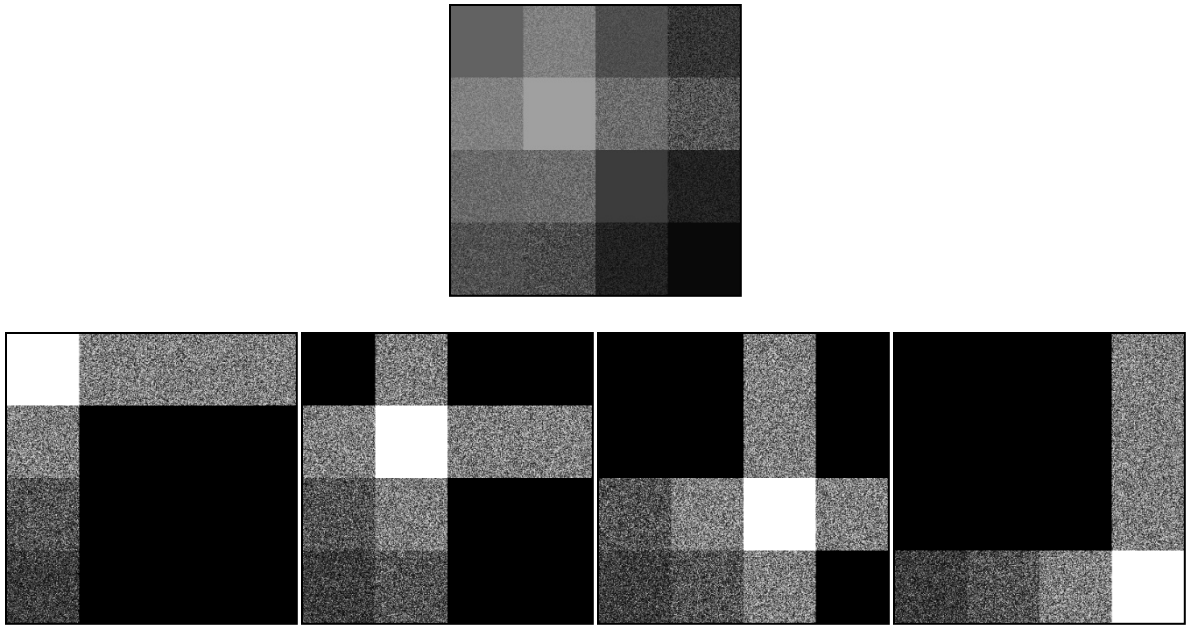
$$((M^T M)^{-1} M^T Y)^T = Y^T ((M^T M)^{-1} M^T)^T = Y^T M ((M^T M)^{-1})^T$$

Με αυτό τον τρόπο οι πολλαπλασιασμοί γίνονται ανάμεσα σε πίνακες, πιο βολικούς για τις δομές του Spark, βελτιώνοντας την ταχύτητα και την πολυπλοκότητα της εφαρμογής. Αρχικά πραγματοποιήθηκε ο υπολογισμός $M^T M$ με χρήση της συνάρτησης computeGramianMatrix της MLlib και στην συνέχεια έγινε η αντιστροφή του με χρήση της βιβλιοθήκης Breeze. Επιπλέον, για τον υπολογισμό του ανάστροφου πίνακα χρησιμοποιήθηκαν RDDs, ενώ οι πολλαπλασιασμοί πραγματοποιήθηκαν με τη βοήθεια της βιβλιοθήκης γραμμικής άλγεβρας της MLlib μέσω των κατανεμημένων δομών της.

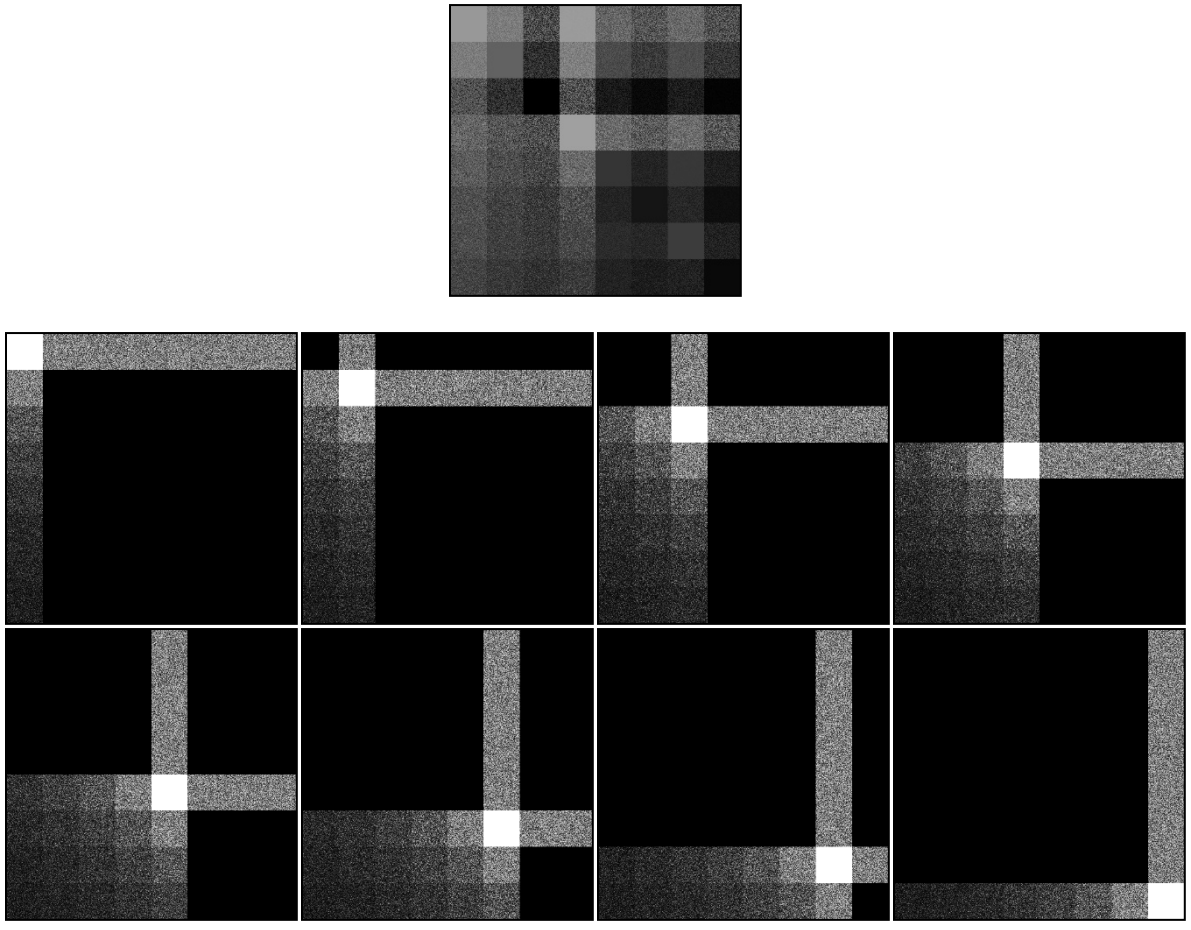
4.5.2. Επαλήθευση Μεθόδου Least Squares

Για την επαλήθευση του αλγορίθμου υπολογίστηκαν τα abundances και σχεδιάστηκαν τα abundance maps για κάθε endmember, δηλαδή εικόνες που παρουσιάζουν την αναλογία του κάθε endmember στα μικτά pixels. Πραγματοποιήθηκαν δύο πειράματα, ένα με συνθετική εικόνα τεσσάρων υλικών και ένα με συνθετική εικόνα οχτώ υλικών.

Παρακάτω παρουσιάζονται τα αποτελέσματα των δύο πειραμάτων, για τέσσερα endmembers(Εικόνα 45) και οχτώ endmembers(Εικόνα 46). Και στις δύο περιπτώσεις είναι φανερό ότι ο αλγόριθμος βρίσκει ορθά τόσο τα σημεία που βρίσκονται τα καθαρά pixels, όσο και τις αναλογίες των endmembers στα υπόλοιπα pixels. Όπως έχει ήδη αναφερθεί η εικόνα χωρίζεται σε κομμάτια και στη διαγώνιό της βρίσκονται p σύνολα από pixels, τα οποία έχουν 100% ποσοστό από ένα μόνο endmember κάθε φορά. Τα pixels αυτά ονομάζονται pure pixels καθώς απαρτίζονται μόνο από ένα υλικό. Εφόσον οι εικόνες εκτυπώνονται σε grayscale το 100% ποσοστό από ένα υλικό εκφράζεται ως το απόλυτο λευκό, ενώ οποιοδήποτε μικρότερο ποσοστό θα εμφανιστεί ως κάποια απόχρωση του γκρι. Το απόλυτο μαύρο αναπαριστά τα μηδενικά ποσοστά ενός υλικού στην εικόνα. Επομένως, παρατηρώντας το πρώτο abundance map του πρώτου πειράματος, φαίνεται ότι τα λευκά pixels που βρίσκονται στη θέση 0,0 του μεγάλου τετραγώνου(4x4) αποτελούνται μόνο από ένα υλικό, και ποσοστά αυτού του υλικού βρίσκονται μόνο στην πρώτη γραμμή και στην πρώτη στήλη της εικόνας. Τέλος, τα αποτελέσματα που προέκυψαν από τον αλγόριθμο συμφωνούν απόλυτα με τα ποσοστά τα οποία χρησιμοποιήθηκαν εξαρχής για την κατασκευή των συνθετικών εικόνων.



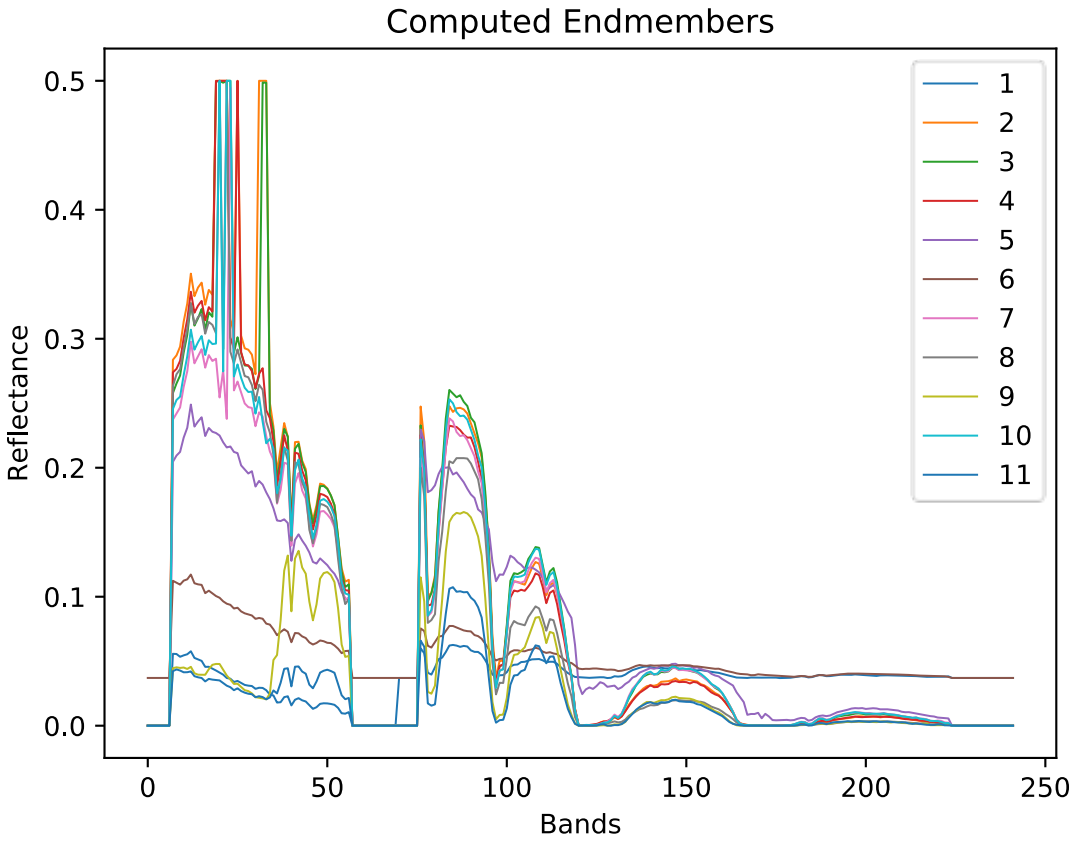
Εικόνα 45 - Πείραμα 1: Abundance Maps



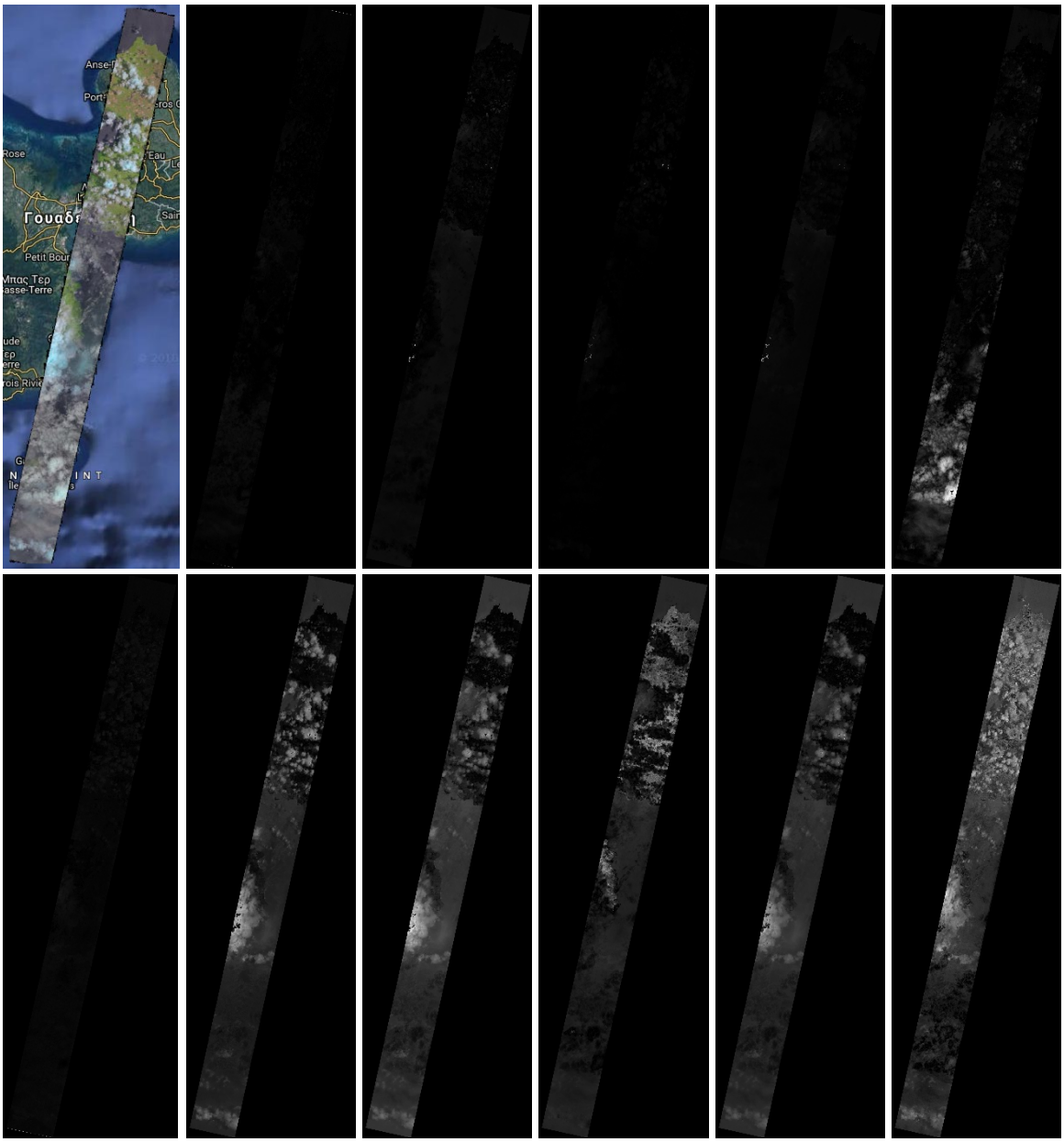
Εικόνα 46 - Πείραμα 2: Abundance Maps

4.6. Spectral Unmixing Πραγματικών Δεδομένων

Μετά τη διαδικασία επαλήθευσης ολόκληρης της εφαρμογής που αναπτύχθηκε, πραγματοποιήθηκε δοκιμή με είσοδο πραγματικών δεδομένων. Χρησιμοποιήθηκε η υπερφασματική απεικόνιση της Γουαδελούπης όπως σαρώθηκε από το εργαλείο Hyperion του δορυφόρου EO-1 (Εικόνα 48-πάνω αριστερά). Ο αλγόριθμος εντόπισε έντεκα υλικά στην εικόνα και εξήγαγε έντεκα abundance maps αντίστοιχα. Παρακάτω παρουσιάζονται οι φασματικές υπογραφές(Εικόνα 47) των υλικών που εντοπίστηκαν και τα abundance maps τους(Εικόνα 48).



Εικόνα 47 - Φασματικές υπογραφές πραγματικής εικόνας



Εικόνα 48 - Abundance maps πραγματικής εικόνας

ΚΕΦΑΛΑΙΟ 5 – ΑΝΑΛΥΣΗ ΕΠΙΔΟΣΗΣ ΕΦΑΡΜΟΓΗΣ SPARK

Το κεφάλαιο αυτό έχει σκοπό να αναδείξει το τρόπο με τον οποίο επηρεάζεται η επίδοση μίας εφαρμογής Spark τόσο από παραμετροποιήσεις στο λογισμικό όσο και παραμετροποιήσεις στην αρχιτεκτονική του συστήματος. Η βελτιστοποίηση της επίδοσης βασίζεται αρχικά σε αλλαγές στον κώδικα χρήστη και με βάση αυτές εφαρμόζονται αλλαγές και στη διαχείριση των πόρων του cluster.

5.1. Πλήθος Partitions

Ο αριθμός των partitions στα οποία χωρίζεται η βασική δομή δεδομένων του Spark(RDD) αποτελεί ένα πολύ σημαντικό παράγοντα για την ταχύτητα εκτέλεσης μίας εφαρμογής. Ο χρήστης έχει τη δυνατότητα να ορίσει τον αριθμό των partitions μέσω της παραμέτρου spark.default.parallelism, η οποία έχει ως προεπιλεγμένη τιμή το συνολικό πλήθος των cores όλων των executors του cluster. Στη βιβλιογραφία προτείνεται η παράμετρος αυτή να τίθεται ίση με το πλήθος των cores επί δύο ή τρεις φορές ως βέλτιστη επιλογή. Επιπλέον, η ελάχιστη τιμή της δε θα πρέπει να είναι μικρότερη από το συνολικό πλήθος των cores, καθώς σε αυτή την περίπτωση κάποια cores δε θα χρησιμοποιούνται. Αυτό συμβαίνει, διότι κάθε core επεξεργάζεται ένα partition κάθε στιγμή, με αποτέλεσμα αν έχουν οριστεί λιγότερα partitions, η επεξεργασία ενός RDD να γίνεται από λιγότερα cores. Τέλος, ο υπερβολικός αριθμός από partitions μειώνει την επίδοση της εφαρμογής, καθώς ο πολύ μεγάλος αριθμός από μικρά partitions δυσχεραίνει τη διαχείρισή τους από το Spark.

Στην παρούσα διπλωματική εργασία πραγματοποιήθηκαν δύο πειράματα, ένα με συνθετική εικόνα τεσσάρων υλικών και ένα με συνθετική εικόνα οχτώ υλικών. Σε κάθε πείραμα δοκιμάστηκαν διάφορες τιμές της παραμέτρου spark.default.parallelism, ώστε να γίνει αντιληπτός ο τρόπος με τον οποίο επηρεάζει ο αριθμός των partitions την επίδοση μίας εφαρμογής Spark.

Στο πρώτο πείραμα, πραγματοποιήθηκαν δοκιμές με 12, 24, 36, 60, 120, 242 και 1000 partitions. Για κάθε αριθμό από partitions πραγματοποιήθηκαν περισσότερες από μία εκτελέσεις, ώστε η μέτρηση του χρόνο ολοκλήρωσης της εφαρμογής να υπολογιστεί ως μέσος όρος των τιμών. Παρακάτω παρουσιάζονται οι αναλυτικές μετρήσεις χρόνου που συλλέχθηκαν από το Web UI του Spark για το πρώτο πείραμα.

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019185336-0004	Unmixing	12	2.0 GB	2018/10/19 18:53:36	spark	FINISHED	2.7 min
app-20181019184937-0003	Unmixing	12	2.0 GB	2018/10/19 18:49:37	spark	FINISHED	2.7 min
app-20181019184559-0002	Unmixing	12	2.0 GB	2018/10/19 18:45:59	spark	FINISHED	2.8 min
app-20181019184233-0001	Unmixing	12	2.0 GB	2018/10/19 18:42:33	spark	FINISHED	2.7 min
app-20181019183813-0000	Unmixing	12	2.0 GB	2018/10/19 18:38:13	spark	FINISHED	2.7 min

Εικόνα 49 - 12 Partitions - 4 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019192138-0004	Unmixing	12	2.0 GB	2018/10/19 19:21:38	spark	FINISHED	2.0 min
app-20181019191758-0003	Unmixing	12	2.0 GB	2018/10/19 19:17:58	spark	FINISHED	2.0 min
app-20181019191324-0002	Unmixing	12	2.0 GB	2018/10/19 19:13:24	spark	FINISHED	2.0 min
app-20181019191021-0001	Unmixing	12	2.0 GB	2018/10/19 19:10:21	spark	FINISHED	2.0 min
app-20181019190752-0000	Unmixing	12	2.0 GB	2018/10/19 19:07:52	spark	FINISHED	2.0 min

Εικόνα 50 - 24 Partitions - 4 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019182330-0004	Unmixing	12	2.0 GB	2018/10/19 18:23:30	spark	FINISHED	2.0 min
app-20181019182055-0003	Unmixing	12	2.0 GB	2018/10/19 18:20:55	spark	FINISHED	2.0 min
app-20181019181758-0002	Unmixing	12	2.0 GB	2018/10/19 18:17:58	spark	FINISHED	2.0 min
app-20181019181334-0001	Unmixing	12	2.0 GB	2018/10/19 18:13:34	spark	FINISHED	2.0 min
app-20181019180519-0000	Unmixing	12	2.0 GB	2018/10/19 18:05:19	spark	FINISHED	2.0 min

Εικόνα 51 - 36 Partitions - 4 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019194821-0004	Unmixing	12	2.0 GB	2018/10/19 19:48:21	spark	FINISHED	2.1 min
app-20181019194544-0003	Unmixing	12	2.0 GB	2018/10/19 19:45:44	spark	FINISHED	2.1 min
app-20181019194320-0002	Unmixing	12	2.0 GB	2018/10/19 19:43:20	spark	FINISHED	2.1 min
app-20181019194058-0001	Unmixing	12	2.0 GB	2018/10/19 19:40:58	spark	FINISHED	2.1 min
app-20181019193708-0000	Unmixing	12	2.0 GB	2018/10/19 19:37:08	spark	FINISHED	2.1 min

Εικόνα 52 - 60 Partitions - 4 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019201756-0004	Unmixing	12	2.0 GB	2018/10/19 20:17:56	spark	FINISHED	2.3 min
app-20181019201405-0003	Unmixing	12	2.0 GB	2018/10/19 20:14:05	spark	FINISHED	2.3 min
app-20181019201100-0002	Unmixing	12	2.0 GB	2018/10/19 20:11:00	spark	FINISHED	2.3 min
app-20181019200657-0001	Unmixing	12	2.0 GB	2018/10/19 20:06:57	spark	FINISHED	2.3 min
app-20181019200356-0000	Unmixing	12	2.0 GB	2018/10/19 20:03:56	spark	FINISHED	2.3 min

Εικόνα 53 - 120 Partitions - 4 Endmembers

Completed Applications (8)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019210032-0007	Unmixing	12	2.0 GB	2018/10/19 21:00:32	spark	FINISHED	3.3 min
app-20181019205640-0006	Unmixing	12	2.0 GB	2018/10/19 20:56:40	spark	FINISHED	3.4 min
app-20181019205307-0005	Unmixing	12	2.0 GB	2018/10/19 20:53:07	spark	FINISHED	3.3 min
app-20181019204855-0004	Unmixing	12	2.0 GB	2018/10/19 20:48:55	spark	FINISHED	3.3 min
app-20181019204455-0003	Unmixing	12	2.0 GB	2018/10/19 20:44:55	spark	FINISHED	3.6 min
app-20181019204109-0002	Unmixing	12	2.0 GB	2018/10/19 20:41:09	spark	FINISHED	3.5 min
app-20181019203730-0001	Unmixing	12	2.0 GB	2018/10/19 20:37:30	spark	FINISHED	3.4 min
app-20181019203402-0000	Unmixing	12	2.0 GB	2018/10/19 20:34:02	spark	FINISHED	3.2 min

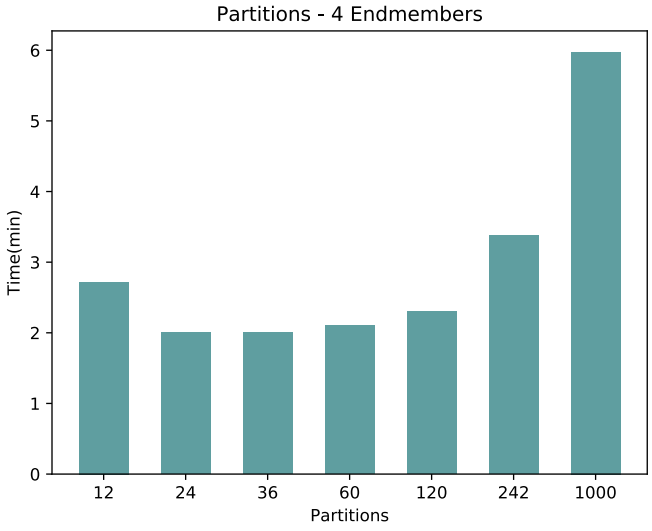
Εικόνα 54 - 242 Partitions - 4 Endmembers

Completed Applications (8)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181019232224-0007	Unmixing	12	2.0 GB	2018/10/19 23:22:24	spark	FINISHED	5.7 min
app-20181019215545-0006	Unmixing	12	2.0 GB	2018/10/19 21:55:45	spark	FINISHED	6.0 min
app-20181019214918-0005	Unmixing	12	2.0 GB	2018/10/19 21:49:18	spark	FINISHED	5.8 min
app-20181019214230-0004	Unmixing	12	2.0 GB	2018/10/19 21:42:30	spark	FINISHED	5.8 min
app-20181019213542-0003	Unmixing	12	2.0 GB	2018/10/19 21:35:42	spark	FINISHED	6.2 min
app-20181019212932-0002	Unmixing	12	2.0 GB	2018/10/19 21:29:32	spark	FINISHED	5.9 min
app-20181019212249-0001	Unmixing	12	2.0 GB	2018/10/19 21:22:49	spark	FINISHED	6.5 min
app-20181019211626-0000	Unmixing	12	2.0 GB	2018/10/19 21:16:26	spark	FINISHED	5.9 min

Εικόνα 55 - 1000 Partitions - 4 Endmembers

Για την καλύτερη κατανόηση των παραπάνω στοιχείων σχεδιάστηκε το ιστόγραμμα της Εικόνας 56 , στην οποία απεικονίζονται οι μέσες τιμές των χρόνων εκτέλεσης ανά πλήθος partitions.



Εικόνα 56 - Ιστόγραμμα: Partitions - 4 Endmembers

Παρατηρώντας το παραπάνω ιστόγραμμα επιβεβαιώνεται ότι η επιλογή που προτείνεται από τη βιβλιογραφία, όπου στην περίπτωση αυτή είναι τα 24 ή 36 partitions, προσφέρει τη βέλτιστη επίδοση στην εφαρμογή. Επιπλέον, η επιλογή των 12 partitions εμφανίζει χειρότερο χρόνο εκτέλεσης, καθώς το μέγεθος των partitions είναι μεγαλύτερο και τα tasks αργούν να ολοκληρωθούν. Τέλος, ο υπερβολικά μεγάλος αριθμός από partitions(1000) εμφανίζει έντονη αύξηση του χρόνου ολοκλήρωσης της εφαρμογής, διότι τεμαχίζει τα δεδομένα σε πολύ μικρά κομμάτια.

Στο δεύτερο πείραμα, πραγματοποιήθηκαν δοκιμές με 120, 242, 500 και 1000 partitions. Οι δοκιμές των 12 , 24 ,36 και 60 δεν υλοποιήθηκαν, διότι το πλήθος των partitions έπρεπε να είναι μεγαλύτερο εξαιτίας του μεγέθους των δεδομένων αυτού του πειράματος. Ομοίως με το προηγούμενο πείραμα, πραγματοποιήθηκαν περισσότερες από μία εκτελέσεις, με σκοπό να υπολογιστεί ο μέσος όρος των τιμών. Παρακάτω παρουσιάζονται οι αναλυτικές μετρήσεις χρόνου που συλλέχθηκαν από το Web UI του Spark για το δεύτερο πείραμα.

Completed Applications (8)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181020202953-0007	Unmixing	12	2.0 GB	2018/10/20 20:29:53	spark	FINISHED	7.2 min
app-20181020202212-0006	Unmixing	12	2.0 GB	2018/10/20 20:22:12	spark	FINISHED	7.1 min
app-20181020201255-0005	Unmixing	12	2.0 GB	2018/10/20 20:12:55	spark	FINISHED	7.0 min
app-20181020200522-0004	Unmixing	12	2.0 GB	2018/10/20 20:05:22	spark	FINISHED	6.9 min
app-20181020195711-0003	Unmixing	12	2.0 GB	2018/10/20 19:57:11	spark	FINISHED	7.7 min
app-20181020194629-0002	Unmixing	12	2.0 GB	2018/10/20 19:46:29	spark	FINISHED	8.6 min
app-20181020193837-0001	Unmixing	12	2.0 GB	2018/10/20 19:38:37	spark	FINISHED	6.9 min
app-20181020193016-0000	Unmixing	12	2.0 GB	2018/10/20 19:30:16	spark	FINISHED	7.5 min

Εικόνα 57 - 120 Partitions - 8 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181020212907-0004	Unmixing	12	2.0 GB	2018/10/20 21:29:07	spark	FINISHED	8.6 min
app-20181020212017-0003	Unmixing	12	2.0 GB	2018/10/20 21:20:17	spark	FINISHED	8.4 min
app-20181020211114-0002	Unmixing	12	2.0 GB	2018/10/20 21:11:14	spark	FINISHED	8.6 min
app-20181020210158-0001	Unmixing	12	2.0 GB	2018/10/20 21:01:58	spark	FINISHED	8.7 min
app-20181020205001-0000	Unmixing	12	2.0 GB	2018/10/20 20:50:01	spark	FINISHED	8.7 min

Εικόνα 58 - 242 Partitions - 8 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181020230157-0004	Unmixing	12	2.0 GB	2018/10/20 23:01:57	spark	FINISHED	10 min
app-20181020225018-0003	Unmixing	12	2.0 GB	2018/10/20 22:50:18	spark	FINISHED	11 min
app-20181020222412-0002	Unmixing	12	2.0 GB	2018/10/20 22:24:12	spark	FINISHED	12 min
app-20181020221143-0001	Unmixing	12	2.0 GB	2018/10/20 22:11:43	spark	FINISHED	12 min
app-20181020215952-0000	Unmixing	12	2.0 GB	2018/10/20 21:59:52	spark	FINISHED	11 min

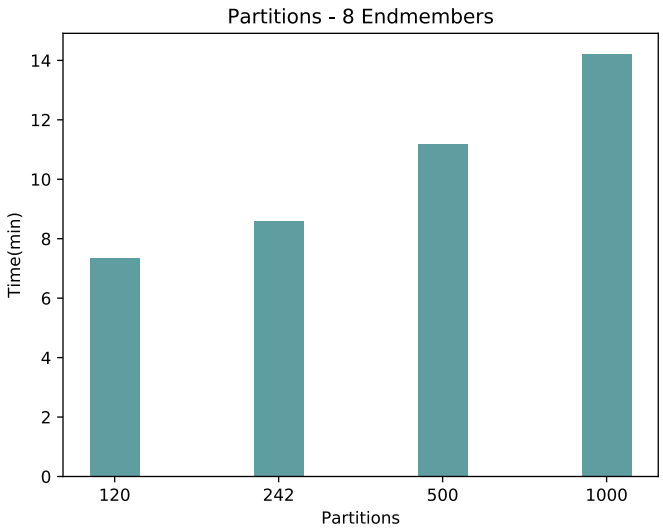
Εικόνα 59 - 500 Partitions - 8 Endmembers

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021003135-0004	Unmixing	12	2.0 GB	2018/10/21 00:31:35	spark	FINISHED	13 min
app-20181021001438-0003	Unmixing	12	2.0 GB	2018/10/21 00:14:38	spark	FINISHED	14 min
app-20181020235338-0002	Unmixing	12	2.0 GB	2018/10/20 23:53:38	spark	FINISHED	16 min
app-20181020233825-0001	Unmixing	12	2.0 GB	2018/10/20 23:38:25	spark	FINISHED	14 min
app-20181020232345-0000	Unmixing	12	2.0 GB	2018/10/20 23:23:45	spark	FINISHED	14 min

Εικόνα 60 - 1000 Partitions - 8 Endmembers

Στο ιστόγραμμα της Εικόνας 61 απεικονίζονται οι μέσες τιμές των χρόνων εκτέλεσης ανά πλήθος partitions. Με βάση όσων αναφέρθηκαν παραπάνω, είναι εμφανές και εδώ ότι όσο αυξάνονται τα partitions τόσο χειροτερεύει η επίδοση της εφαρμογής.



Εικόνα 61 - Ιστόγραμμα: Partitions - 8 Endmembers

Συνοψίζοντας, αποδεικνύεται από τα παραπάνω πειράματα ότι βέλτιστες επιλογές είναι τα 36 partitions για την εικόνα με τα τέσσερα endmembers και τα 120 partitions για την εικόνα με τα οχτώ endmembers.

5.2. Μέγεθος Μνήμης Executor

Η επίδοση μίας εφαρμογής Spark επηρεάζεται τόσο από τον αριθμό των partitions όσο και από τους πόρους που προσφέρει το cluster. Το πλήθος των cores επηρεάζει σε σημαντικό βαθμό την επίδοση και προτείνεται να είναι όσο το δυνατόν πιο υψηλό, σε περίπτωση που δε χρειάζεται να εκτελεστούν παράλληλα με το Spark στο cluster και άλλου είδους εφαρμογές. Στην παρούσα διπλωματική εργασία επιλέχθηκε ο αριθμός των cores να είναι δώδεκα, δηλαδή όσα ακριβώς προσφέρονται. Επομένως, αυτό που χρειάζεται να εξετασθεί ακόμη είναι το κατά πόσο το μέγεθος της μνήμης που διαχειρίζεται ο κάθε executor επηρεάζει την επίδοση της εφαρμογής.

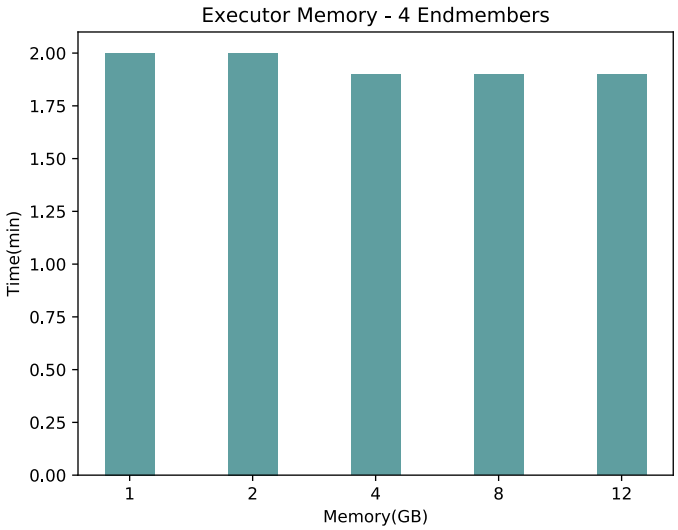
Πραγματοποιήθηκαν, λοιπόν, δύο πειράματα σχετικά με το μέγεθος της μνήμης που χρειάζεται ο κάθε executor. Το πρώτο πείραμα έγινε με συνθετική εικόνα τεσσάρων υλικών και το δεύτερο πείραμα με εικόνα οχτώ υλικών. Και στα δύο πειράματα έγιναν δοκιμές με 1GB, 2Gb, 4GB, 8GB και 12GB μνήμης ανά executor.

Παρακάτω παρουσιάζονται οι αναλυτικές μετρήσεις χρόνου που συλλέχθηκαν από το Web UI του Spark, καθώς και το ιστόγραμμα χρόνου για το πρώτο πείραμα.

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021121800-0004	Unmixing	12	12.0 GB	2018/10/21 12:18:00	spark	FINISHED	1.9 min
app-20181021121540-0003	Unmixing	12	8.0 GB	2018/10/21 12:15:40	spark	FINISHED	1.9 min
app-20181021121312-0002	Unmixing	12	4.0 GB	2018/10/21 12:13:12	spark	FINISHED	1.9 min
app-20181021121050-0001	Unmixing	12	2.0 GB	2018/10/21 12:10:50	spark	FINISHED	2.0 min
app-20181021120826-0000	Unmixing	12	1024.0 MB	2018/10/21 12:08:26	spark	FINISHED	2.0 min

Εικόνα 62 - Executor Memory - 4 Endmembers



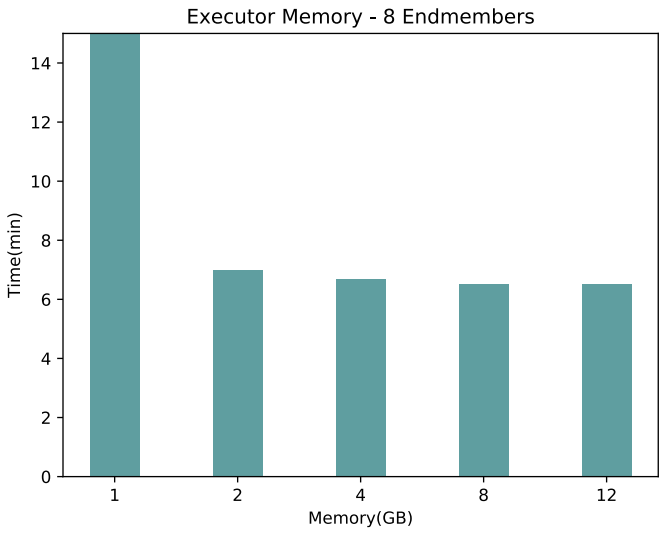
Εικόνα 63 - Ιστόγραμμα: Executor Memory - 4 Endmembers

Στην συνέχεια φαίνονται οι μετρήσεις και το ιστόγραμμα χρόνου και του δεύτερου πειράματος. Το ιστόγραμμα εμφανίζεται μεγεθυμένο ως προς τον άξονα του χρόνου, διότι η μέτρηση για το 1GB μνήμης είναι πολύ μεγάλη(39min).

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021132337-0004	Unmixing	12	12.0 GB	2018/10/21 13:23:37	spark	FINISHED	6.5 min
app-20181021131624-0003	Unmixing	12	8.0 GB	2018/10/21 13:16:24	spark	FINISHED	6.5 min
app-20181021130857-0002	Unmixing	12	4.0 GB	2018/10/21 13:08:57	spark	FINISHED	6.7 min
app-20181021130105-0001	Unmixing	12	2.0 GB	2018/10/21 13:01:05	spark	FINISHED	7.0 min
app-20181021122106-0000	Unmixing	12	1024.0 MB	2018/10/21 12:21:06	spark	FINISHED	39 min

Εικόνα 64 - Executor Memory - 8 Endmembers



Εικόνα 65 - Ιστόγραμμα: Executor Memory - 8 Endmembers

Παρατηρώντας τα αποτελέσματα και των δύο πειραμάτων, είναι φανερό ότι η επίδοση της εφαρμογής Spark βελτιώνεται όσο αυξάνεται το μέγεθος της μνήμης κάθε executor. Αξίζει, να σημειωθεί, όμως, ότι το cluster που χρησιμοποιήθηκε αποτελούνταν από μηχανήματα συγκεκριμένης μνήμης, επομένως δεν πραγματοποιήθηκαν πειράματα με υπερβολικά μεγάλο μέγεθος μνήμης ανά executor. Στην περίπτωση υπερβολικής μνήμης, με βάση την πηγή [28], εμφανίζονται έντονες καθυστερήσεις στις λειτουργίες του garbage collector, με αποτέλεσμα να τίθεται ως ασφαλές μέγιστο όριο μνήμης, ανά executor, τα 64GB.

5.3. Πλήθος Executors

Ένας ακόμη παράγοντας που αξίζει να εξετασθεί ως αναφορά την επίδοση μίας εφαρμογής Spark είναι το συνολικό πλήθος των executors που διαχειρίζεται ο driver. Ο τρόπος με τον οποίο ορίζεται το πλήθος των executors αλλάζει με βάση τον cluster manager που χρησιμοποιείται. Στην περίπτωση της παρούσας διπλωματικής εργασίας, ο Standalone προσφέρει την επιλογή στο χρήστη να αλλάξει την παραπάνω ρύθμιση μέσω της παραμέτρου spark.executor.cores. Η παράμετρος αυτή προσδιορίζει ουσιαστικά τον αριθμό των cores που προσφέρονται σε κάθε executor, ορίζοντας έμμεσα το πόσοι executors θα σηκωθούν σε κάθε worker, με βάση τα cores που διαθέτει.

Το cluster που χρησιμοποιήθηκε στα πειράματα αυτής της εργασίας, προσφέρει συγκεκριμένους συνδυασμούς πλήθους executors με μέγεθος μνήμης ανά executor. Στον παρακάτω πίνακα φαίνονται όλοι οι δυνατοί συνδυασμοί:

Πλήθος Executors	*Μνήμη ανά Executor*
2	1GB έως 12GB
3	1GB έως 6GB
6	1GB έως 3GB
12	1GB

Πραγματοποιήθηκαν δύο πειράματα, ένα με συνθετική εικόνα τεσσάρων υλικών και ένα με συνθετική εικόνα οχτώ υλικών. Και στα δύο πειράματα επιλέχθηκε να εξετασθούν οι συνδυασμοί που περιλαμβάνουν τη μέγιστη μνήμη ανά executor, δηλαδή 2 executors-12GB, 3 executors-6GB, 6 executors-3GB και 12 executors-1GB.

Παρακάτω εμφανίζονται οι αναλυτικές πληροφορίες σχετικά με τον αριθμό των executors και για τα δύο πειράματα, όπως παρουσιάζονται στο Web UI του Spark.

Executor Summary (2)

ExecutorID	Worker	Cores	Memory	State	Logs
1	worker-20181021170239-147.102.19.81-44965	4	12288	RUNNING	stdout stderr
0	worker-20181021170240-147.102.19.78-35425	8	12288	RUNNING	stdout stderr

Executor Summary (3)

ExecutorID	Worker	Cores	Memory	State	Logs
2	worker-20181021170239-147.102.19.81-44965	4	6144	RUNNING	stdout stderr
1	worker-20181021170240-147.102.19.78-35425	4	6144	RUNNING	stdout stderr
0	worker-20181021170240-147.102.19.78-35425	4	6144	RUNNING	stdout stderr

Εικόνα 66 - Πλήθος και μνήμη Executors (2ex-12GB, 3ex-6GB)

Executor Summary (6)

ExecutorID	Worker	Cores	Memory	State	Logs
1	worker-20181021170240-147.102.19.78-35425	2	3072	RUNNING	stdout stderr
5	worker-20181021170239-147.102.19.81-44965	2	3072	RUNNING	stdout stderr
2	worker-20181021170240-147.102.19.78-35425	2	3072	RUNNING	stdout stderr
4	worker-20181021170239-147.102.19.81-44965	2	3072	RUNNING	stdout stderr
3	worker-20181021170240-147.102.19.78-35425	2	3072	RUNNING	stdout stderr
0	worker-20181021170240-147.102.19.78-35425	2	3072	RUNNING	stdout stderr

Executor Summary (12)

ExecutorID	Worker	Cores	Memory	State	Logs
7	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
8	worker-20181021170239-147.102.19.81-44965	1	1024	RUNNING	stdout stderr
1	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
0	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
5	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
4	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
2	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
9	worker-20181021170239-147.102.19.81-44965	1	1024	RUNNING	stdout stderr
10	worker-20181021170239-147.102.19.81-44965	1	1024	RUNNING	stdout stderr
6	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
3	worker-20181021170240-147.102.19.78-35425	1	1024	RUNNING	stdout stderr
11	worker-20181021170239-147.102.19.81-44965	1	1024	RUNNING	stdout stderr

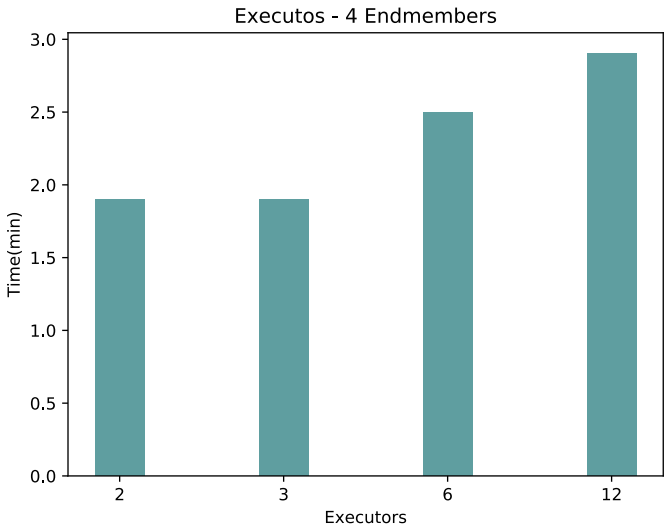
Εικόνα 67 - Πλήθος και μνήμη Executors (6ex-3GB, 12ex-1GB)

Το πείραμα με την συνθετική εικόνα των τεσσάρων υλικών έδωσε τα αποτελέσματα που εμφανίζονται στην Εικόνα 68 και στο ιστόγραμμα χρόνου.

Completed Applications (4)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021171137-0003	Unmixing	12	1024.0 MB	2018/10/21 17:11:37	spark	FINISHED	2.9 min
app-20181021170841-0002	Unmixing	12	3.0 GB	2018/10/21 17:08:41	spark	FINISHED	2.5 min
app-20181021170555-0001	Unmixing	12	6.0 GB	2018/10/21 17:05:55	spark	FINISHED	1.9 min
app-20181021170301-0000	Unmixing	12	12.0 GB	2018/10/21 17:03:01	spark	FINISHED	1.9 min

Εικόνα 68 - Executors - 4 Endmembers



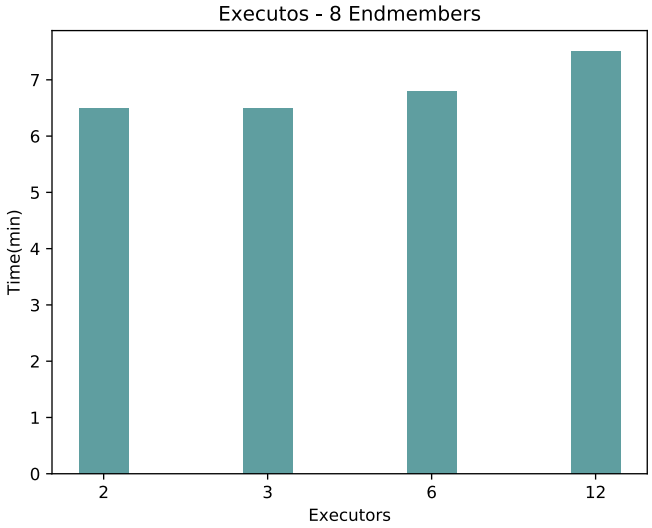
Εικόνα 69 - Ιστόγραμμα: Executors - 4 Endmembers

Στη συνέχεια εμφανίζονται τα αποτελέσματα που προέκυψαν από το δεύτερο πείραμα, που πραγματοποιήθηκε με την συνθετική εικόνα των οχτώ υλικών.

Completed Applications (6)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021175859-0003	Unmixing	12	1024.0 MB	2018/10/21 17:58:59	spark	FINISHED	7.5 min
app-20181021175020-0002	Unmixing	12	3.0 GB	2018/10/21 17:50:20	spark	FINISHED	6.8 min
app-20181021174246-0001	Unmixing	12	6.0 GB	2018/10/21 17:42:46	spark	FINISHED	6.5 min
app-20181021172053-0000	Unmixing	12	12.0 GB	2018/10/21 17:20:53	spark	FINISHED	6.5 min

Εικόνα 70 - Executors - 8 Endmembers



Εικόνα 71 - Ιστόγραμμα: Executors - 8 Endmembers

Συνοψίζοντας, τα αποτελέσματα των δύο πειραμάτων δείχνουν ότι το πλήθος των executors πρέπει να παραμένει χαμηλό, καθώς όσο αυξάνεται, η επίδοση της εφαρμογής αρχίζει να μειώνεται. Με βάση την πηγή [29], μεγαλύτερος αριθμός από executors ισοδυναμεί με μεγαλύτερο αριθμό από JVMs και με περισσότερα αντίγραφα των δεδομένων, στις περιπτώσεις που χρησιμοποιούνται broadcast μεταβλητές.

5.4. Κλιμάκωση Δεδομένων

Στο σημείο αυτό της εργασίας, θεωρήθηκε χρήσιμο να πραγματοποιηθούν μετρήσεις σχετικά με το τρόπο που επηρεάζεται η επίδοση της εφαρμογής από την κλιμάκωση των δεδομένων εισόδου. Τα πειράματα που διεξήχθησαν παρουσιάζουν το ποσοστό χρόνου εκτέλεσης των τεσσάρων βασικών σταδίων της εφαρμογής ως προς το συνολικό χρόνο ολοκλήρωσής της. Η εφαρμογή που υλοποιήθηκε αποτελείται από το στάδιο ανάγνωσης των δεδομένων, από τον αλγόριθμο HFC-VD, από το Endmember Extraction και από το Abundance Estimation.

Πραγματοποιήθηκαν δοκιμές με πέντε συνθετικές εικόνες και μία πραγματική. Η πρώτη εικόνα αποτελείται από τέσσερα καθαρά υλικά και έχει 160.000 pixels, η δεύτερη από πέντε με 250.000 pixels, η τρίτη από έξι με 360.000 pixels, η τέταρτη από επτά με 490.000 pixels, η πέμπτη από οχτώ με 640.000 pixels και τέλος η πραγματική εικόνα με 3.025.131 pixels. Παρακάτω παρατίθενται η Εικόνα 72 με τους χρόνους εκτέλεσης της εφαρμογής στις πέντε περιπτώσεις συνθετικών εικόνων, η Εικόνα 73 με το χρόνο εκτέλεσης της πραγματικής και ο πίνακας των ποσοστών χρόνου του κάθε σταδίου σε σχέση με το συνολικό χρόνο εκτέλεσης για κάθε εικόνα.

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181021200017-0004	Unmixing	12	12.0 GB	2018/10/21 20:00:17	spark	FINISHED	6.5 min
app-20181021195319-0003	Unmixing	12	12.0 GB	2018/10/21 19:53:19	spark	FINISHED	4.9 min
app-20181021194833-0002	Unmixing	12	12.0 GB	2018/10/21 19:48:33	spark	FINISHED	3.7 min
app-20181021194511-0001	Unmixing	12	12.0 GB	2018/10/21 19:45:11	spark	FINISHED	2.7 min
app-20181021194201-0000	Unmixing	12	12.0 GB	2018/10/21 19:42:01	spark	FINISHED	2.0 min

Εικόνα 72 - Total time συνθετικών εικόνων

Completed Applications (1)

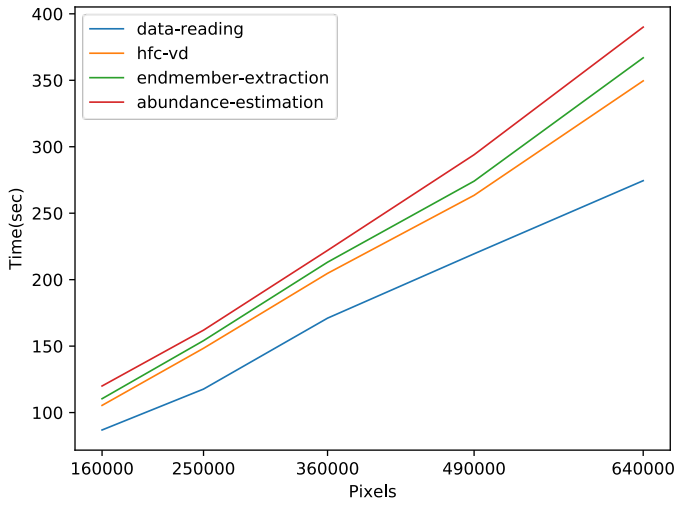
Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20181028121854-0015	Unmixing	9	6.0 GB	2018/10/28 12:18:54	spark	FINISHED	35 min

Εικόνα 73 - Total time πραγματικής εικόνας

	Data Reading	HFC-VD	Endmember Extraction	Abundance Estimation	Total Time (min)
Image 1	72.415%	15.416%	4.1875%	7.9815%	2.0
Image 2	72.638%	18.95%	3.537%	4.875%	2.7
Image 3	77.04%	15.18%	3.782%	3.998%	3.7
Image 4	74.643%	14.965%	3.624%	6.768%	4.9
Image 5	70.375%	19.25%	4.445%	5.93%	6.5
Real Image	62.483%	19.638%	10.876%	7.003%	35

Όπως προκύπτει από τις μετρήσεις του πίνακα, το ποσοστό του χρόνου που αναλογεί σε κάθε στάδιο της εφαρμογής παραμένει σχετικά σταθερό καθώς αυξάνεται ο όγκος των δεδομένων. Ιδιαίτερη εντύπωση προκαλεί το υψηλό ποσοστό του σταδίου ανάγνωσης των δεδομένων, το οποίο είναι εμφανές ότι καθυστερεί την εφαρμογή. Η επίδοση των υπόλοιπων σταδίων της εφαρμογής είναι σε πολύ ικανοποιητικά επίπεδα, καθώς ο χρόνος εκτέλεσης και των τριών αλγορίθμων είναι πολύ μικρός.

Εν συνεχεία, σχεδιάστηκε ένα γράφημα που παρουσιάζει συγκριτικά τον τρόπο με τον οποίο αυξάνεται ο χρόνος εκτέλεσης κάθε σταδίου της εφαρμογής για τις εικόνες του πειράματος.



Εικόνα 74 - Συγκριτικό διάγραμμα σταδίων

Με βάση το παραπάνω διάγραμμα, γίνεται αντιληπτό ότι η κλίση και των τεσσάρων συναρτήσεων είναι παρόμοια, πράγμα που σημαίνει ότι ο χρόνος ολοκλήρωσης κάθε σταδίου της εφαρμογής αυξάνεται με ίδιο τρόπο καθώς μεγαλώνουν τα δεδομένα.

ΚΕΦΑΛΑΙΟ 6 - ΕΠΙΛΟΓΟΣ

6.1. Αξιολόγηση Υλοποίησης

Στην παρούσα διπλωματική εργασία, υλοποιήθηκε μία εφαρμογή Spark με σκοπό την ανάλυση υπερφασματικών δεδομένων μέσω της μεθόδου Linear Spectral Unmixing. Η πλατφόρμα στην οποία σχεδιάστηκε η εφαρμογή περιλαμβάνει το εργαλείο Apache Spark, το κατανεμημένο σύστημα αρχείων HDFS, τον Standalone cluster manager του Spark και τις βιβλιοθήκες MLlib, Breeze και Geotrellis. Επιπλέον, για το πειραματικό μέρος της εργασίας χρησιμοποιήθηκε ένα cluster τριών υπολογιστών συγκεκριμένων δυνατοτήτων.

Οι μετρήσεις που προέκυψαν παρουσιάζουν πληροφορίες σχετικά με τον τρόπο που επηρεάζεται η επίδοση της εφαρμογής από το πλήθος των partitions, το μέγεθος της μνήμης ανά executor, το συνολικό πλήθος των executors, αλλά και την κλιμάκωση των δεδομένων. Αρχικά, διαπιστώθηκε ότι ο χρόνος εκτέλεσης της εφαρμογής είναι βέλτιστος όταν χρησιμοποιείται πλήθος partitions ίσο με το τριπλάσιο του αριθμού των συνολικών cores του συστήματος. Όταν αυξάνεται το μέγεθος των δεδομένων εισόδου, όμως, είναι πολύ πιθανό το πλήθος των partitions να χρειαστεί να αυξηθεί. Εν συνεχεία, η εφαρμογή εκτελείται βέλτιστα με χρήση της μέγιστης δυνατής μνήμης ανά executor, ενώ χρησιμοποιεί το ελάχιστο πλήθος από executors που μπορεί να σηκώσει το cluster. Τέλος, η κλιμάκωση των δεδομένων επηρεάζει με ανάλογο τρόπο κάθε ένα στάδιο της εφαρμογής, κλιμακώνοντας χρονικά την εκτέλεσή της.

Οι επιλογές των επιμέρους συστημάτων της πλατφόρμας αποδείχθηκαν κατάλληλες τόσο για την πολυπλοκότητα της εφαρμογής όσο και για το μέγεθος των δεδομένων εισόδου. Το εργαλείο Apache Spark προσέφερε τη δυνατότητα εκτέλεσης της εφαρμογής σε ένα κατανεμημένο σύστημα, με στόχο την εύκολη διαχείριση μεγάλου όγκου δεδομένων σε μικρό χρόνο. Οι βιβλιοθήκες που υποστηρίζονται από το Spark, όπως η MLlib και η Breeze, που χρησιμοποιήθηκαν στην εργασία αυτή, ήταν απαραίτητες για τον υπολογισμό αλγεβρικών πράξεων μεταξύ μεγάλων πινάκων καθώς και για τη χρήση αλγορίθμων μηχανικής μάθησης. Αξίζει να σημειωθεί η αναγκαιότητα της βιβλιοθήκης Geotrellis, η οποία ανέλαβε τη διαδικασία εισαγωγής των δεδομένων, τα οποία είναι σε τύπο αρχείων GeoTIFF, γεγονός που δυσκόλευε την ανάγνωσή τους από τις δομές του Spark. Τέλος, η επιλογή του HDFS, διασφάλισε την αξιόπιστη αποθήκευση και το διαμοιρασμό των δεδομένων μεταξύ του cluster.

6.2. Μελλοντικές Επεκτάσεις

Η εφαρμογή που υλοποιήθηκε στην παρούσα διπλωματική εργασία αποτελεί μία ικανοποιητική λύση για το πρόβλημα του Linear Spectral Unmixing. Τα αποτελέσματα και η επίδοση της ήταν σε πολύ καλό βαθμό δεδομένου των πόρων που προσέφερε το cluster μηχανημάτων που χρησιμοποιήθηκε. Παρόλα αυτά, κρίνεται πως το cluster περιλάμβανε μηχανήματα με οριακά μεγέθη μνήμης και CPU για την ομαλή επεξεργασία μεγάλων υπερφασματικών δεδομένων. Επομένως, για την εξαγωγή συμπερασμάτων όσων αφορά την επίδοση της εφαρμογής σε μεγάλα δεδομένα, θα ήταν προτιμότερο να χρησιμοποιηθούν μηχανήματα με περισσότερους πόρους.

Παρατηρώντας τα αποτελέσματα του πειραματικού μέρους, είναι εμφανές ότι η ανάγνωση των δεδομένων καταναλώνει το μεγαλύτερο μέρος του χρόνου εκτέλεσης της εφαρμογής. Με σκοπό τη μείωση του χρόνου αυτού, προτείνεται η αλλαγή του τύπου των δεδομένων εισόδου ώστε να εξυπηρετεί την ανάγνωσή τους από το Spark. Πιο συγκεκριμένα, μία υλοποίηση η οποία θα δέχεται ως είσοδο ένα Multispectral GeoTIFF αρχείο με ενσωματωμένα όλα τα κανάλια του υπερφασματικού κύβου, αντί για 242 διαφορετικά Singlespectral GeoTIFF αρχεία, θα μείωνε κατά πολύ το χρόνο εκτέλεσης των αλγορίθμων.

Συνοψίζοντας, η εφαρμογή που σχεδιάστηκε και υλοποιήθηκε θα μπορούσε να ανταπεξέλθει σε ρεαλιστικές συνθήκες για την εξαγωγή αποτελεσμάτων σχετικά με τα endmembers και τα abundances μίας υπερφασματικής απεικόνισης. Εντούτοις, κρίνεται απαραίτητο τα δεδομένα εισόδου να δεχθούν την κατάλληλη προεπεξεργασία για να συλλεχθούν ακριβέστερα αποτελέσματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Alexander F.H. Goetz. (2011). Measuring the Earth from Above: 30 years (and Counting) of Hyperspectral Imaging
https://www.photonics.com/Article.aspx?AID=47298

[2] Wikipedia. Hyperspectral imaging
https://en.wikipedia.org/wiki/Hyperspectral_imaging#cite_note-lu-4

[3] Vinay Kumar. Hyperspectral Remote Sensing
https://nrsc.gov.in/sites/all/pdf/SPIE%20APRS%20Tutorial_Hyperspectral%20RS_Vinay%20Kumar.pdf

[4] Gabriel Martin Hernandez. (2013). Design and implementation of new methods for spatial preprocessing prior to spectral unmixing of remotely sensed hyperspectral data

[5] Sanchez, Sergio & Martín, Gabriel & Plaza, Antonio & Chang, Chein-I. (2010). GPU Implementation of Fully Constrained Linear Spectral Unmixing for Remotely Sensed Hyperspectral Data Exploitation. Proceedings of SPIE - The International Society for Optical Engineering. 7810. 10.1117/12.860775.

[6] Zebin Wu, Member, IEEE, Yonglong Li, Antonio Plaza, Fellow, IEEE, Jun Li, Member, IEEE, Fu Xiao, Member, IEEE, and Zhihui Wei (2016). Parallel and Distributed Dimensionality Reduction of Hyperspectral Data on Cloud Computing Architectures

[7] Turing Finance. (2014). Dimensionality Reduction Techniques
http://www.turingfinance.com/artificial-intelligence-and-statistics-principal-component-analysis-and-self-organizing-maps/

[8] Jakob Sigurðsson. (2015). Hyperspectral Unmixing Using Total Variation and Sparse Methods

[9] Mauro Dalla Mura, Jocelyn Chanussot, Antonio Plaza (2014). An Overview on Hyperspectral Unmixing
http://www-ljk.imag.fr/membres/Faouzi.Triki/projetPbsInverses/Pre/DallaMura_unmixing.pdf

[10] Wikipedia. Apache Spark
https://en.wikipedia.org/wiki/Apache_Spark

[11] Apache Spark™ - Unified Analytics Engine for Big Data
https://spark.apache.org/

[12] Apache Spark Documentation. Submitting Applications
https://spark.apache.org/docs/latest/submitting-applications.html

[13] Jacek Laskowski. Mastering Apache Spark
https://jaceklaskowski.gitbooks.io/mastering-apache-spark/

[14] Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia. (2015). Learning Spark Lightning-fast Big Data Analysis. O'Reilly Media

[15] TechVidvan Team. (2018). Apache Spark Partitioning and Spark Partition
https://techvidvan.com/tutorials/spark-partition/

[16] Oracle. (2014). ODI 12c - Spark SQL and Hive?
https://blogs.oracle.com/dataintegration/odi-12c-spark-sql-and-hive

[17] Joseph K. Bradley. (2015). Spark DataFrames and ML Pipelines
https://www.slideshare.net/databricks/dataframes-and-pipelines

[18] Apache Spark. GraphX
https://spark.apache.org/graphx/

[19] Data Flair. (2017). Directed Acyclic Graph DAG in Apache Spark
https://data-flair.training/blogs/dag-in-apache-spark/

[20] Alexey Grishchenko. (2016). Distributed Systems Architecture
https://0x0fff.com/spark-memory-management/

[21] USGS. Spectral Library Version 7. Hyperion
https://crustal.usgs.gov/speclab/HYPRN.php

[22] edureka!. (2018). Apache Hadoop HDFS Architecture
https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

[23] Emanuele Torti, Alessandro Fontanella, Antonio Plaza. (2017). Parallel real-time virtual dimensionality estimation for hyperspectral images

[24] Chein-I Chang. (2018). A Review of Virtual Dimensionality for Hyperspectral Imagery

[25] C. Andreou, V. Karathanassi. (2011). Using Principal Component Analysis for Endmember Extraction

[26] C.-I Chang, C. Wu, W. Liu and Y.C. Ouyang. 2006. A growing method for simplex-based endmember extraction algorithms

[27] Wikipedia. Least squares
https://en.wikipedia.org/wiki/Least_squares`

[28] Cloudera. Tuning Spark Applications
https://www.cloudera.com/documentation/enterprise/5-9-x/topics/admin_spark_tuning.html

[29] Henry Hu. (2017). How We Optimise Apache Spark Jobs
https://rea.tech/how-we-optimize-apache-spark-apps/

ΠΑΡΑΡΤΗΜΑ

Α. Υλοποίηση της εφαρμογής σε γλώσσα προγραμματισμού Scala

Παρακάτω παρουσιάζεται ο κώδικας που αναπτύχθηκε για την υλοποίηση της εφαρμογής:

```
package unmixing

/**Java imports**/
import java.net.URI

/**Geotrellis imports**/
import geotrellis.raster.io.geotiff.reader.GeoTiffReader
import geotrellis.spark.io.hadoop._

/**Scala imports**/
import scala.math._
import scala.collection.mutable.ListBuffer
import scala.util.control._

/**Spark imports**/
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.rdd.RDD
import org.apache.spark.mllib.linalg.Vector
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.distributed.{IndexedRow,
IndexedRowMatrix, RowMatrix}
import org.apache.spark.storage.StorageLevel

/**Hadoop imports**/
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.fs.FileUtil

/**Math imports**/
import breeze.numerics.{sqrt}
import breeze.linalg._
import org.apache.commons.math3.distribution.NormalDistribution


object Main {

  def main(args: Array[String]) {

    /**Spark Configuration**/

    val conf = new SparkConf()
    conf.setMaster("spark://147.102.19.80:7077")
    conf.setAppName("Unmixing")
    conf.set("spark.executor.memory", "6g")
```

```
    conf.set("spark.executor.cores", "3")
    val sc = new SparkContext(conf)

    /**Read HDFS**/

    val dir: String = "hdfs://147.102.19.80:9000/Input/"
    val configuration = new Configuration()
    val hdfs = FileSystem.get(URI.create(dir), configuration)
    val fileStatus = hdfs.listStatus(dir)
    val paths = FileUtil.stat2Paths(fileStatus)
    val list = paths.toList.sortBy(_.toString)

    val N =
HadoopGeoTiffReader.readSingleband(list(0))(sc).tile.toArrayDouble.size
    val L = list.length

    val rdd = sc.union(
      list.map(path => sc.parallelize(Seq((list.indexOf(path).toLong,
Vectors.dense(HadoopGeoTiffReader.readSingleband(path)(sc).tile.toArrayDouble
))))))

    val rdd_T = rowToColumnStore(rdd)
    rdd_T.persist(StorageLevel.MEMORY_AND_DISK)
    val matT = new RowMatrix(rdd_T)

    /**VD**/

    val cor_ar = matT.computeGramianMatrix().toArray.map(x => x/N)
    val cor = breeze.linalg.DenseMatrix.create(L,L,cor_ar)

    val cov_ar = matT.computeCovariance().toArray
    val cov = breeze.linalg.DenseMatrix.create(L,L,cov_ar)

    val es_cor = eigSym(cor)
    val es_cov = eigSym(cov)

    val l_cor = sc.parallelize(es_cor.eigenvalues.toArray)
    val l_cov = sc.parallelize(es_cov.eigenvalues.toArray)

    val diff = l_cor.zip(l_cov).map(x => x._1+(-x._2))
    val variance = l_cor.map(x => x*x).zip(l_cov.map(x => x*x)).map(x =>
x._1+x._2).map(x => sqrt(x*2/N))

    val infNorm5 = variance.map(x => new NormalDistribution(0,
x).inverseCumulativeProbability(0.00001))
    val vd = diff.zip(infNorm5).filter(x => round(x._1*10000)/10000 > -
round(x._1*10000)/10000).count()


sc.parallelize(Seq(vd)).saveAsTextFile("hdfs://147.102.19.80:9000/output/vd")

    /**PCA**/

    val pc = matT.computePrincipalComponents(vd.toInt-1)
    val pca: RowMatrix = matT.multiply(pc)
    val pca_index = pca.rows.zipWithIndex().map(_.swap)
```

```
    val pca_rddT = rowToColumnStore(pca_index).cache()
    val max = pca_rddT.map(x => x.toArray.indexOf(x.toArray.max))
    val min = pca_rddT.map(x => x.toArray.indexOf(x.toArray.min))

    val index = max.union(min).distinct
    val pot_end = sc.parallelize(index.collect.toList.map(x =>
pca_index.lookup(x).head)).zipWithIndex().map(_.swap)
    val pot_endT = rowToColumnStore(pot_end)

    val cosine = new RowMatrix(pot_endT).columnSimilarities()
    val num_pot_end = cosine.numCols()
    pca_rddT.unpersist()

    /**Discard**/

    var keep = new ListBuffer[Long]()
    var discard = new ListBuffer[Long]()
    var degree = 1

    val loop = new Breaks

    while (discard.length < (num_pot_end-vd)){
      val temp = cosine.entries.filter(x => cos(Pi*degree/180) < x.value &&
x.value< cos(Pi*(degree-1)/180))

      val slist = temp.collect.toList.sortBy(_.value)

      loop.breakable{
        for (x <- slist){
          if (keep.contains(x.i)){
            if (keep.contains(x.j)){
              keep-= x.j
              discard+= x.j
            }
            else if (!discard.contains(x.j)) discard+= x.j
          }
          else if(discard.contains(x.i)){
            if (!keep.contains(x.j) && !discard.contains(x.j)) keep+= x.j
          }
          else{
            if (keep.contains(x.j)) discard+= x.i
            else if (discard.contains(x.j)) keep+= x.i
            else{
              keep+= x.i
              discard+= x.j
            }
          }
          if (discard.length == (num_pot_end-vd)) loop.break
        }
      }
      degree+= 1
    }

    /**Abundance Estimation**/

    val t = index.collect()
    val l = discard.map(_.toInt).map(x => t(x))
```

```
    val end = t.toList diff l

    val r = matT.rows.map(x => Vectors.dense(x.toArray.map(_/65535)))

    val M_T = sc.parallelize(end.map(x =>
r.zipWithIndex().map(_.swap).lookup(x).head).filter(_.numNonzeros >
100)).zipWithIndex().map(_.swap)
    M_T.saveAsTextFile("hdfs://147.102.19.80:9000/output/endmembers")

    val M = new RowMatrix(rowToColumnStore(M_T))
    val gram = M.computeGramianMatrix().toArray
    val dimension = sqrt(gram.length).toInt
    val inv_gram =
inv(breeze.linalg.DenseMatrix.create(dimension,dimension,gram, 0, dimension,
true))
    val M_breeze = breeze.linalg.DenseMatrix.create(L,dimension,
M.rows.map(_.toArray).collect.flatten, 0, dimension, true)
    val mul = M_breeze * inv_gram.t
    val mul_local = new org.apache.spark.mllib.linalg.DenseMatrix(L,
dimension, mul.toArray)
    val a = new RowMatrix(r).multiply(mul_local)
    a.rows.saveAsTextFile("hdfs://147.102.19.80:9000/output/abundance")

    rdd_T.unpersist()
    sc.stop()
  }

  def rowToColumnStore(data: RDD[(Long, Vector)]): RDD[Vector]={
    data.flatMap(x =>
x._2.toArray.zipWithIndex.zip(Array.fill(x._2.size)(x._1)))
      .map {case ((a, b), c) => (b, (c, a))}
      .groupByKey
      .sortByKey()
      .map(x => Vectors.dense(x._2.toList.sortBy(_._1).map(_._2).toArray))
  }
}
```

Β. Αρχείο build.sbt

Ο κώδικας του αρχείου build.sbt που χρησιμοποιήθηκε:

```
name := "Unmixing_D"

version := "0.1"

scalaVersion := "2.11.6"

libraryDependencies ++= {
  val sparkVer = "2.3.0"
  Seq(
    "org.apache.spark" %% "spark-core" % sparkVer % "provided" withSources(),
    "org.apache.spark" %% "spark-mllib" % sparkVer % "provided"
withSources(),
    "org.locationtech.geotrellis" %% "geotrellis-spark" % "1.2.0"
  )
}
```

Γ. Κώδικας κατασκευής συνθετικών δεδομένων

Παρακάτω παρατίθεται ο κώδικας που αναπτύχθηκε σε γλώσσα προγραμματισμού Python για την κατασκευή των συνθετικών εικόνων:

```
import numpy as np
import os
import glob
from libtiff import TIFF
import math

##Read list of files

file_list = glob.glob('./Sample/*.txt')
endmembers = []

for file in file_list:
  d = np.loadtxt(file, skiprows = 1)
  endmembers.append(d)

##Create output folders

if not os.path.exists('./Images'):
    os.makedirs('./Images')
if not os.path.exists('./Text'):
    os.makedirs('./Text')

value = 10000
pixels = np.empty(len(endmembers)*len(endmembers)*value)

##Create abundances

cut = int(math.sqrt(len(pixels))/len(endmembers))
abu = np.empty([int(np.sqrt(len(pixels))), int(np.sqrt(len(pixels))),
len(endmembers)])

for i in range(0, len(endmembers)):
      if (i == len(endmembers) - 1):
            i_index = int(math.sqrt(len(pixels)))
      else:
            i_index = (i+1)*cut
      for x in range(i*cut, i_index):
            for y in range(i*cut, i_index):
                  abu[x][y] = np.zeros(len(endmembers))
                  abu[x][y][i] = 1

for i in range(0, len(endmembers)):
      for j in range(i+1, len(endmembers)):
            if(i == len(endmembers) - 1):
                  i_index = int(math.sqrt(len(pixels)))
            else:
                  i_index = (i+1)*cut

            if(j == len(endmembers) - 1):
                  j_index = int(math.sqrt(len(pixels)))
```

```
            else:
                  j_index = (j+1)*cut
            for x in range(i*cut, i_index):
                  for y in range(j*cut, j_index):
                        abu[x][y] = np.zeros(len(endmembers))
                        rand = np.random.dirichlet(np.ones(2),size=1)
                        while rand[0][0] > 0.95 or rand[0][1] > 0.95:
                              rand = np.random.dirichlet(np.ones(2),size=1)
                        abu[x][y][i] = rand[0][0]
                        abu[x][y][j] = rand[0][1]

for j in range(0, len(endmembers)):
      for i in range(j+1, len(endmembers)):
            if(i == len(endmembers) - 1):
                  i_index = int(math.sqrt(len(pixels)))
            else:
                  i_index = (i+1)*cut
            if(j == len(endmembers) - 1):
                  j_index = int(math.sqrt(len(pixels)))
            else:
                  j_index = (j+1)*cut
            for x in range(i*cut, i_index):
                  for y in range(j*cut, j_index):
                        abu[x][y] = np.zeros(len(endmembers))
                        rand = np.random.dirichlet(np.ones(i+1-j),size=1)
                        while i+1-j == 2 and (rand[0][0] > 0.95 or rand[0][1]
> 0.95):
                              rand = np.random.dirichlet(np.ones(i+1-
j),size=1)
                        for k in range(j, i+1):
                              abu[x][y][k] = rand[0][k-j]

abun = np.reshape(abu, (len(pixels), len(endmembers)))
np.savetxt("./Text/abu.txt", abun)

##Create one image per band

for i in range(0, len(endmembers[0])):
      for j in range(0, len(pixels)):
            sum = 0.0
            for k in range(0, len(endmembers)):
                  if endmembers[k][i] == -1.2300000000000000043e+34:
                        sum = sum
                  else:
                        sum = sum + endmembers[k][i]*abun[j][k]*65535

            pixels[j] = round(sum)

      out = np.reshape(pixels.astype(np.uint16),
(int(math.sqrt(len(pixels))), int(math.sqrt(len(pixels)))))
      np.savetxt("./Text/test%03d.txt" %i, out)
      tiff = TIFF.open("./Images/test%03d.TIF" %i, "w");
      tiff.write_image(out)
      TIFF.close(tiff);
```