ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση του αλγορίθμου συστάσεων SVD στην πλατφόρμα Spark

Αλέξανδρος Ιωαννίδης

Επιβλέποντες:

Ηρακλής Βαρλάμης, Επίκουρος Καθηγητής

Δημήτριος Μιχαήλ,Επίκουρος Καθηγητής

Κωνσταντίνος Τσερπές, Λέκτορας

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΣΥΣΤΑΣΕΩΝ SVD ΣΤΗΝ ΠΛΑΤΦΟΡΜΑ SPARK

Αλέξανδρος Ιωαννίδης

Α.Μ.: 21208

Επιβλέποντες: Ηρακλής Βαρλάμης, Επίκουρος Καθηγητής

Στον πατέρα μου Θεόφιλο,

στην μητέρα μου Μόνικα,

στην αδελφή μου Ελένη,

και στους φίλους μου Μανόλη και Σωκράτη

Περιεχόμενα

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία με τίτλο «Εξατομικευμένη παροχή συστάσεων με βάση το περιεχόμενο και τις προτιμήσεις» διενεργήθηκε στο πλαίσιο της υποχρεωτικής εκπόνησης πτυχιακής εργασίας στο κτίριο που στεγάζεται το τμήμα Πληροφορικής και Τηλεματικής του Χαροκοπείου Πανεπιστημίου Αθηνών.

Η διεκπεραίωση της πτυχιακής εργασίας επιτεύχθει με την αναλυτική καθοδήγηση και στενή παρακολούθηση του Επίκουρου Καθηγητή Βαρλάμη Ηρακλή και τον οποίο θα ήθελα να ευχαριστήσω για όλες τις άμεσες και επεξηγηματικές απαντήσεις του σε όλες τις απορίες μου.

1. ΕΙΣΑΓΩΓΗ

1.1 Πεδίο διατριβής

Το θέμα της διατριβής, αφορά την επέκταση ενός συστήματος στηριζόμενο σε λογισμικό, το οποίο προτρέπει χρήστες μιας ιστοσελίδας εικονικής κοινότητας, με συμβουλευτικές προτάσεις για τη παρακολούθηση νέων ταινιών προσαρμοσμένες στις ανάγκες του κάθε χρήστη ξεχωριστά, όπως το γνωστό MovieLens.

Η δημιουργία των προτάσεων, προκύπτει από την συσσώρευση πληροφοριών, από το περιεχόμενο κάθε χρήστη και την ιστορική καταγραφή των προτιμήσεών του, σύμφωνα με τη συμμετοχή του σε προηγούμενες ψηφοφορίες στις οποίες έχει κληθεί να βαθμολογήσει ταινίες. Το σύστημα αναπτύχθηκε με τη μέθοδο συνεργατικού φιλτραρίσματος Collaborative Filtering (CF) με την στενότερη έννοια, δηλαδή δημιουργώντας αυτόματες προβλέψεις για τους χρήστες. Μια από τις μεθόδους που υλοποιείται το (CF) και στην οποία βασίζεται το λογισμικό που έχει αναπτυχθεί, είναι η υβριδική μέθοδος που συνδυάζει δύο τεχνικές πρώτον με βάση το περιεχόμενο του χρήστη (content based) και δεύτερον το (item based) το οποίο είναι το ιστορικό των προτιμήσεων κάθε χρήστη.

Το ήδη υπάρχον λογισμικό που πρόκειται να επεκταθεί σε άλλη γλώσσα προγραμματισμού από μέρους μου, παρουσιάζει έναν αλγόριθμο που υλοποιεί την προηγούμενη υβριδική μέθοδο. Ο αλγόριθμος αυτός είναι ο Singular Value Decomposition (SVD).

Οι συστάσεις που γίνονται από το σύστημα στο χρήστη, προκύπτουν με βάση την επεξεργασία των αποθηκευμένων στοιχείων που αφορούν τον κάθε χρήστη και από τον αλγόριθμο (SVD) ο οποίος υπολογίζει και εξάγει με τη παραγοντοποίηση ενός πολύπλοκου πίνακα ο οποίος περιέχει στατιστικά στοιχεία με βάση τα οποία γίνονται οι προβλέψεις για τη πιθανή μελλοντική συμπεριφορά ενός χρήστη.

1.2 Σκοπός

Στόχος της παρούσας πτυχιακής εργασίας, είναι η επέκταση υπάρχουσας εφαρμογής με την συμβολή της βιβλιοθήκης Lenskit. Σε αυτό το σκοπό συμπεριλαμβάνονται τα ακόλουθα στάδια, πρώτα η παραμετροποίηση του αρχικού σειριακού κώδικα δοσμένου σε C++ έτσι ώστε να δέχεται ως είσοδο τα dataset του MovieLens, έπειτα μετατροπή του ήδη υπάρχοντος προγράμματος σειριακής ακολουθίας, κατασκευασμένου σε γλώσσα προγραμματισμού C++ σε σειριακό πρόγραμμα, με χρήση της γλώσσας προγραμματισμού Java, και έπειτα η μετατροπή του σειριακού κώδικα ανεπτυγμένου σε Java σε παράλληλο πρόγραμμα χρησιμοποιώντας την πλατφόρμα ανάπτυξης Apache Spark η οποία αποτελεί μια γρήγορη μηχανή επεξεργασίας δεδομένων μεγάλης κλίμακας και της γλώσσας προγραμματισμού Java.

Τελικά, γίνεται η χρονική καταγραφή της εκτέλεσης του σειριακού προγράμματος αναπτυγμένου σε Java και του αντίστοιχου παράλληλου προγράμματος επίσης σε Java έτσι ώστε να διαπιστωθεί αν παράγονται παρόμοια αποτελέσματα και να γίνει αντιπαραβολή των διαφορετικών χρόνων εκτέλεσης των προγραμμάτων και να παρατηρηθεί αν έχει επιτευχθεί κάποια βελτιστοποίηση με τη παραλληλοποίηση ή όχι, και συμπερασματικά αν υπάρχουν περιθώρια περαιτέρω βελτίωσης του προγράμματος.

1.3 Σύνοψη αποτελεσμάτων

Συνοπτικά, τα αποτελέσματα που προέκυψαν ήταν αρχικά από την παραμετροποιήση του αρχικού κώδικα σε C++, δηλαδή η εξαγωγή των αποτελεσμάτων για το dataset του MovieLens των 100000 βαθμολογιών, έπειτα με την εκτέλεση του σειριακού κώδικα που μετατράπηκε σε Java διαπιστώθηκε αξιοσημείωτη βελτίωση στον συνολικό χρόνο υλοποίησης καθώς επίσης και στους χρόνους εκτέλεσης των επιμέρους μεθόδων συγκριτικά με τον αρχικό κώδικα σε C++. Η διάρκεια για την κατασκευή του engine, του load history και του processing test είναι 0 seconds, άρα συνολικά 6 seconds, χρησιμοποιώντας ως training file το u.data το οποίο αποτελεί ολόκληρο το data set αυτό των 100.000 βαθμολογιών όπου και οι χρήστες και οι ταινίες είναι απαριθμημένα συνεκτικά από το 1. Ως testing file χρησιμοποιήθηκε το u1.test το οποίο αποτελείται από 2 κομμάτια, το πρώτο κομμάτι αποτελεί το 80% του συνόλου και είναι από το u data και το δεύτερο κομμάτι αποτελεί το 20% του συνόλου και είναι από το test data. Τέλος χρησιμοποιήθηκε το u1.predictions αντίστοιχα για να γραφτούν τα παραγόμενα αποτελέσματα με μέση εκτίμηση λάθους να είναι 0,679812.

Στην συνέχεια το παράλληλο πρόγραμμα υλοποιημένο σε Java στο Spark επιτυγχάνει την παραλληλοποιήση όλων των function εκτός της CalcFeatures η οποία εκτελείται ακολουθιακά όπως προηγουμένως. Συγκεκριμένα ο χρόνος υλοποιήσης του engine construction διαρκεί 5 seconds, της LoadHistory 3 seconds και η ProcessTest 1 second, ενώ η εκτέλεση για τη CalcFeatures που εκτελείται σειριακά είναι 18 seconds. Συνεπώς ο συνολικός χρόνος εκτέλεσης του παράλληλου προγράμματος είναι 27 seconds χρησιμοποιώντας το MoovieLens Dataset ίδιου μεγέθους με αυτό που χρησιμοποιήθηκε στα άλλα 2 προγράμματα δηλαδή αυτό των 100.000 βαθμολογιών.

Η υλοποιήση της function CalcFeatures δυστυχώς παρέμεινε σειριακή λόγω των περιορισμών παραλληλοποιήσης που έχει ο αρχικός αλγόριθμος σε αυτό το κομμάτι του.

Η κυρίως εργασία του αλγορίθμου γίνεται με τη μέθοδο CalcFeatures που σκοπό έχει τον υπολογισμό των τιμών στους 2 πίνακες CustomerFeatures και MovieFeatures. Οι τιμές στα κελιά των πινάκων αυτών ενημερώνονται κατά κύματα (epochs) μέχρι να επιτευχθεί κάποιο αποδεκτό συνολικό σφάλμα. Η εργασία αυτή αποτελείται από 3 ένθετους βρόχους, με κατ΄ ελάχιστο εκτελέσεις των εντολών του εσωτερικού βρόγχου να είναι $[10^5 * 120 * 64 = 768000000]$.

Οι εντολές αυτές είναι κυρίως οι 4 βασικές πράξεις αριθμητικής και ανάγνωση και εγγραφή σε 2 δισδιάστατους και έναν μονοδιάστατο πίνακα.

Επιπρόσθετα όλοι οι πίνακες έχουν μετατραπεί σε κάποιο RDD<Type>, οποιαδήποτε διάβασμα αλλά και τροποποίηση δεδομένων απαιτεί κάποιο transformation ή action πάνω στο RDD. Για τα transformation χρειάζεται να περιγραφεί η μετατροπή με lambda expression, anonymous class, inner class ή static nested class. Για να επιτευχθεί η επιθυμητή λειτουργικότητα R/W σε πολλαπλά RDD σε ένα ενιαίο βήμα, θα έπρεπε να μπορεί να γίνει κλήση RDD transformation/action μέσα από την function του anonymous class/lambda expression, πράγμα που απαγορεύεται στο Spark.

Στην ενότητα 7 με τα συμπεράσματα γίνεται μια αναλυτική περιγραφή με τα αναλυτικά συμπεράσματα, την αδυναμία παραλληλοποίησης ενός συγκεκριμένου τμήματος κώδικα και τις δυσκολίες που παρουσιάστηκαν κατά την διαδικασία εκπόνησης της παρούσας πτυχιακής εργασίας.

2. ΕΠΙΣΤΗΜΟΝΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 Βασικές έννοιες και βασικές βιβλιογραφικές αναφορές

Σημαντικές έννοιες, που εμπεριέχονται στην υπάρχουσα εργασία, είναι αρχικά η έννοια του recommendation system το οποίο αποτελεί μια υποκλάση ενός συστήματος φιλτραρίσματος πληροφοριών που έχει ως σκοπό την αναζήτηση της πρόβλεψης της βαθμολογίας ή της προτίμησης των χρηστών σε αντικείμενα. Δημοφιλείς εφαρμογές στις οποίες χρησιμοποιούνται συνήθως είναι σε ταινίες όπως και στη περίπτωση της τρέχουσας εργασίας αλλά και σε τραγούδια.

Επιπρόσθετα, η έννοια του Collaborative filtering είναι μια τεχνική που χρησιμοποιείται από τη προαναφερθείσα έννοια recommendation system και διακρίνεται σε 2 έννοιες την πιο στενή και την πιο γενική. Νεότερη καθώς επίσης και αυτή που εξετάζεται στην εργασία είναι η στενότερη έννοια στην οποία η διαδικασία του φιλτραρίσματος αφορά τις πληροφορίες των χρηστών ή ακόμα καλύτερα τη τάση που έχουν να συμπεριφέρονται οι χρήστες. Επιπλέον αυτή η τεχνική δημιουργεί αυτόματες προβλέψεις για τα ενδιαφέροντα των χρηστών συλλέγοντας πληροφορίες για τις προτιμήσεις ή τη συσχέστιση της συμπεριφοράς πολλών χρηστών ή και τα δύο μαζί ταυτόχρονα.

Πρόσθετα, σπουδαία κρίνεται η σημασία του αλγορίθμου singular value decomposition ο οποίος υλοποιεί τη παραγοντοποίηση ενός σχήματος UΣV* όπου το U αποτελεί έναν m x m μοναδιαίο πίνακα, το Σ αποτελεί έναν m x n διαγώνιο πίνακα χωρίς αρνητικούς αριθμούς στη κύρια διαγώνιό του και το V* αποτελεί τον συζυγή ενός n x n μοναδιαίου πίνακα.

Επίσης, σημαντική είναι η έννοια του data set που αποτελεί μια συλλογή σχετικών και διακριτών στοιχείων από συσχετιζόμενα δεδομένα που είναι οργανωμένα σε κάποιο είδος δομής δεδομένων που μπορούν να προσπελαστούν ατομικά ή σε συνδυασμό ή διαχειρίζονται στο σύνολο σαν οντότητα. Μια βάση δεδομένων μπορεί να θεωρηθεί ένα σύνολο δεδομένων, όπως και τα υποτμήματα των δεδομένων μέσα σε μια B που συσχετίζονται με ένα συγκεκριμένο είδος πληροφοριών.

Από κάτω, παραθέτονται βιβλιογραφικές αναφορές που ξεχωρίζουν για τη σημαντικότητά τους και τη συνεισφορά τους στη διεκπεραίωση της πτυχιακής εργασίας.

Από τη βιβλιογραφική αναφορά με τον αριθμό [4], χρησιμοποιήθηκε το δείγμα του πηγαίου κώδικα σε γλώσσα προγραμματισμού C++ που υλοποιεί τον αλγόριθμο SVD, ως σημείο αναφοράς για την παρούσα εργασία. Ο προηγούμενος αυτός κώδικας, τροποποιήθηκε στην ίδια γλώσσα προγραμματισμού έτσι ώστε να δέχεται ως είσοδο dataset στη μορφή του MovieLens, έπειτα μετατράπηκε σε πρόγραμμα Java σειριακής ακολουθίας και έπειτα επεκτάθηκε σε παράλληλο πρόγραμμα επίσης αναπτυγμένο σε Java. Η βιβλιογραφική αναφορά με τον αριθμό [5], προχωράει ένα βήμα μακρύτερα και επεξηγεί πιο αναλυτικά τα κομμάτια που περιέχουν μαθηματικές πράξεις και σύμβολα για τον υπολογισμό του SVD βασισμένο στο πηγαίο κώδικα από τη πηγή [4]. Από τη πηγή [7], αξιοποιήθηκαν θεμελιώδεις γνώσεις για τη χρησιμότητα αλλά και για τη κατανόηση της λειτουργίας του διαδικτυακού ιστότοπου Movie Lens καθώς κατά την διαδικασία ανάπτυξης και ελέγχου των αποτελεσμάτων των προγραμμάτων χρησιμοποιούνται τα data sets του Movie Lens. Από τη πηγή [8], βρέθηκαν και χρησιμοποιήθηκαν τα data sets του Movie Lens. Κυρίως χρησιμοποι ήθηκε το data set με τις 100.000 βαθμολογίες. Από τη πηγή [9], αντλήθηκαν πολλές πληροφορίες για τον τρόπο εγκατάστασης και χρήσης της πλατφόρμας Apache Spark σε εφαρμογές με Java και σε περιβάλλον λειτουργικού συστήματος Linux.

2.2 Σχετικές επιστημονικές εργασίες

Κατά τη διάρκεια εκπόνησης της πτυχιακής εργασίας μου, εντόπισα ορισμένες πτυχιακές εργασίες με συναφές αντικείμενο με το δικό μου. Συγκεκριμένα παραθέτονται οι πηγές από [10] έως [13] τις οποίες θεώρησα σημαντικότερες. Από τη πηγή [10] στην οποία παρουσιάζεται μια γενική περιγραφή των recommendation system και των σύγχρονων μεθόδων που τα υλοποιούν και οι οποίες είναι το content-based, collaborative και το hybrid. Ειδική έμφαση δόθηκε στη παράγραφο 2.3.1, 2.3.2 και 2.3.3 όπου περιγράφεται η διαδικασία πρόσθεσης content based χαρακτηριστικών σε collaborative μοντέλα για τη δημιουργία hybrid μεθόδων. Αναλυτικότερα, από τη πηγή [11] η οποία αποτελεί έρευνα της Microsoft τριών ατόμων, στην οποία έκαναν μια αναλυτική σύγκριση διαφόρων μεθόδων για την υλοποίηση του (CF) με 2 διαφορετικές μετρικές 13 αξιολόγησης, η πρώτη χαρακτηρίζεται από ακρίβεια σε ένα σύνολο από ατομικές προβλέψεις ως προς την απόλυτη μεσαία τυπική απόκλιση, ενώ η δεύτερη μετρική υπολογίζει τη χρησιμότητα μιας βαθμολογημένης λίστας προτεινόμενων αντικειμένων. Επίσης μέσα από την έμμεσα διεξοδική παρουσίαση των διαφορών μεταξύ memory based αλγορίθμων και model based μεθόδων κατανόησα καλύτερα τη λειτουργία της πρώτης κατηγορίας που προβλέπει τις βαθμολογίες ενός ενεργού χρήστη βασιζόμενο σε μερικές πληροφορίες σχετικά με τον χρήστη και ενός συνόλου από υπολογίσιμα βάρη από τη βάση δεδομένων των χρηστών συνολικά. Ενώ στη δεύτερη περίπτωση, αντιλήφθηκα τον τρόπο που υπολογίζεται η αναμενόμενη τιμή μιας ψηφοφορίας για αντικείμενα που δεν έχουν παρακολουθηθεί ακόμα από έναν χρήστη, με βάση τα αποτελέσματα προηγούμενων ψηφοφοριών στις οποίες έχει συμμετάσχει ο ίδιος. Στη συνέχεια, η πηγή [12] που αποτελεί διπλωματική εργασία άλλου ατόμου στην οποία συμπεριλαμβάνεται η υλοποίηση του αλγορίθμου SVD για τη μείωση των διαστάσεων ενός διανυσματικού χώρου ως μέρους ενός ευρύτερου προβλήματος και αφαιρώντας τις μειονεκτικές επιπτώσεις της πολλαπλής σημασίας ή συνωνυμίας όρων διατυπώνοντας τις σημαντικές σχέσεις μεταξύ των όρων, βοήθησε αρκετά στη διαμόρφωση μιας πιο σφαιρικής άποψης για την χρησιμότητα του αλγορίθμου. Επιπλέον κατανόησα καλύτερα τη λειτουργία του αλγορίθμου καθώς επίσης και το ευρύ φάσμα εφαρμογών που έχει ο συγκεκριμένος αλγόριθμος. Τελικά, από τη πηγή [13] έγινε σαφέστερη η λειτουργία του SVD, καθώς επίσης παρουσιάζει τρόπους προσέγγισης του αλγορίθμου από προγραμματιστική σκοπιά, πως θα μπορούσαν να αναπαρισταθούν οι SVD πίνακες και πως μπορούν να υπολογιστούν οι τιμές τους και επιπλέον παρουσίασε μια εναλλακτική προσέγγιση του αλγορίθμου που με βοήθησε αρκετά.

2.3 Περιγραφή συστημάτων που χρησιμοποιήθηκαν στη διατριβή

Για την συγγραφή της εργασίας, χρησιμοποιήθηκε το ευέλικτο σύστημα στοιχειοθεσίας κειμένων LaTex, το οποίο μου επέτρεψε να εστιάσω περισσότερο στη λογική δομή του κειμένου, δίνοντας πολλές δυνατότητες σε εμένα, αλλά ίσως το βασικότερό του πλεονέκτημα είναι η γρήγορη ενσωμάτωση εικόνων, πινάκων, cross-references και τμημάτων κώδικα όπως επίσης και η εύκολη επεξεργασία τους. Πιο συγκεκριμένα χρησιμοποιήθηκε το ShareLaTeX το οποίο είναι ένας online επεξεργαστής LaTeX που επιτρέπει την απευθείας σύνδεση και μεταγλώττιση των έργων σε PDF format. Επίσης επειδή είναι εφαρμογή που στηρίζεται σε διακομιστή είχα πρόσβαση στη συγγραφή της παρούσας πτυχιακής εργασίας συνεχώς, ανεξαρτήτως τοποθεσίας μέσω ενός προγράμματος περιήγησης.

Για την υλοποιήση της εργασίας, χρησιμοποιήθηκε αρχικά η πλατφόρμα ανάπτυξης λογισμικού Visual Studio 13 σε Windows 7 για την παραμετροποιήση του αρχικού αλγορίθμου υλοποιημένου σε C++ που δίνεται από τον ιστότοπο http://www.timelydevelopment.com/demos/NetflixPrize.aspx έτσι ώστε να μπορεί να δέχεται ως είσοδο τα dataset του MovieLens. Για την εκτέλεση του προγράμματος σε C++ χρησιμοποιήθηε το Visual Studio 13 της Microsoft όπως προανέφερα, πρόσβαση στο εργαλείο αυτό είχα από από τον υπολογιστή στο χώρο εργασίας μου σε λειτουργικό σύστημα Windows 7 για την εκτέλεση του προγράμματος σε λογισμικό της Microsoft μόνο και όχι από τον προσωπικό υπολογιστή μου που έχει σαν βασικό λειτουργικό σύστημα μόνο Ubuntu 14.04. Τα χαρακτηριστικά του υπολογιστή εργασίας είναι επεξεργαστής Intel(R) Core(TM) i3 CPU 550, 3.20GHz, RAM 4,00 GB , 32-bit αρχιτεκτονική συστήματος.

Με αυτόν τον τρόπο θα εξαχθούν αποτελέσματα από το predictions file έτσι ώστε να χρησιμοποιηθούν στην συνέχεια για επαλήθευση των αποτελεσμάτων που θα προκύψουν από το σειριακό και παράλληλο πρόγραμμα μετέπειτα. Ύστερα αξιοποιήθηκε το Netbeans IDE 8.1 για την μετατροπή του σειριακού κώδικα από C++ σε σειριακό κώδικα σε Java, σε υπολογιστή με βασικό λειτουργικό σύστημα Ubuntu 14.04 και με τα εξής χαρακτηριστικά: επεξεργαστής Intel Core i7-4700MQ CPU 2.40 GHz x 8, 24 GB RAM και αρχιτεκτονική συστήματος 64-bit. Έπειτα χρησιμοποιήθηκε η πλατφόρμα Apache Spark σε συνδυασμό με τη πλατφόρμα ανάπτυξης λογισμικού IntelliJ IDEA 15 επίσης σε λειτουργικό σύστημα Ubuntu 14.04 για την ανάπτυξη της εφαρμογής παράλληλα. Η έκδοση του Apache Spark 1.6.1 που χρησιμοποιήθηκε είναι η Pre-built για το Hadoop 2.6 και μετά. Επίσης κατά τη διαδικασία εγκατάστασης του IntelliJ IDEA έγινε και εγκατάσταση του SBT Plugin το οποίο εργαλείο δίνει τη δυνατότητα παράλληλης εκτέλεσης εργασιών και συμπεριλαμβάνει την παράλληλη εκτέλεση δοκιμών.

Επίσης, για το σχεδιασμό και την υλοποιήση των διαγραμμάτων που θα ακολουθήσουν στη συνέχεια, χρησιμοποιήθηκε το λογισμικό Gliffy το οποίο είναι SaaS προσβάσιμο από παντού μέσω ενός προγράμματος περιήγησης και χρησιμοποιείτε κυρίως για τη δημιουργία UML διαγραμμαάτων και flowcharts καθώς επίσης το πρόγραμμα Dia το οποίο είναι είναι ένα γενικού σκοπού λογισμικό για σχεδίαση διαγραμματών..

Στη συνέχεια, για τη ανάπτυξη της παράλληλης υλοποίησης χρησιμοποίησα το Virtualbox για τη δημιουργία δύο client μηχανημάτων, και τα 2 έχουν λειτουργικό σύστημα Ubuntu 14.04 όπως επίσης και ο server που είναι το φυσικό μηχάνημα μου. Έτσι δημιούργησα ένα network-mounted shared file system.

Στον host-server μηχάνημα μου τροποποιήσα το αρχείο /etc/exports με διακιώματα root και πρόσθεσα αυτή την εντολή /nfs * (rw,sync,no_root_squash,no_subtree_check) έτσι ώστε οτιδήποτε προστεθεί κάτω από το directory του διακομιστή /nfs να είναι προσβάσιμο σε όλους τους πελάτες χωρίς δικαιώματα χρήστη για αυτό προστέθηκε και στις παραμέτρους το no_root_squash.

Έπειτα, από τη μεριά των clients αρκούσε η εντολή mount -t nfs 192.168.56.1:/nfs /nfs όπου το 192.168.1.24 είναι το domain name του server, στη συνέχεια ακολουθεί το directory του server που θα γίνει mount και τέλος ακολουθεί το target directory του client. Επειδή χρησιμοποιούνται μονοπάτια τοπικού συστήματος αποθήκευσης αρχείων, το αρχείο πρέπει να είναι επίσης προσβάσιμο στο ίδιο μονοπάτι σε όλους τους worker κόμβους όπως και στον driver-master.

Επιπλέον, για να μπορέσω να δημιουργήσω αυτή τη συνδεσμολογία χρειάστηκε να τροποποιήσω και να εγκαταστήσω στο Virtualbox ένα Host-Only Network το vboxnet0 για μπορώ να αναπτύσσω στο χώρο εργασίας ή όταν βρισκόμουν εκτός οικίας και πρόσθεσα στα 2 virtual machine που δημιούργησα για να έχουν τον ρόλο των clients, από ένα adapter Host-only στο καθένα. Επιπρόσθετα ενεργοποιήσα από τις επιλογές Promiscuous Mode να είναι Allow All.

Ενώ όταν βρισκόμουν εντός σπιτιού το "set up" γινόταν με bridged mode. Οπότε για εκείνη την περίπτωση εγκατέστησα και 2 bridged adapters έναν για κάθε εικονικό μηχάνημα. Επομένως ανάλογα σε ποιο χώρο βρισκόμουν έκανα εναλλαγή μεταξύ των "adapters" στα εικονικά μηχανήματα.

Συνεχίζοντας, για την εγκατάσταση του Spark σε Standalone Mode σε Cluster χρειάστηκε να τοποθετηθεί μια μεταγλωττισμένη έκδοση του Spark σε κάθε κόμβο του Cluster. Επιπλέον έπρεπε να είναι εγκατεστημένο το openssh server και στα 3 μηχανήματα έτσι ώστε να μπορούν να επικοινωνούν με ασφαλή τρόπο κατά τη διάρκεια της συνεργασίας τους. Μετά ακολούθησε η ενεργοποίηση του SPARK_SSH_FOREGROUND μέσω της εντολής export SPARK_SSH_FOREGROUND = YES για να μην ζητείται password κατά την επικοινωνία με ssh των πόρων μεταξύ τους. Επιπρόσθετα χρειάστηκε να ρυθμιστεί η διεύθυνση του master το οποίο και έγινε μέσω της εντολής export SPARK_MASTER_IP = 192.168.1.24 όταν είχα σύνδεση με host-only ή export SPARK_MASTER_IP = 192.168.56.1 όταν ήμουν συνδεδεμένος με bridged mode. Σημείωση οι προηγούμενες διευθύνσεις δεν ήταν πάντα ίδιες, ήταν οι πιο συχνές όμως, κάποιες φορές διαφέρανε ανάλογα με την IP address που έπαιρνε το host μήχανημα από τον DHCP server. Η εκκίνηση του standalone master server έγινε στο φυσικό μηχάνημα εκτελώντας την εντολή .spark-1.6.1-bin-hadoop2.6/sbin/start-master.sh καθώς η εγκατάσταση του Spark έγινε στο directory κάτω από το /home/alex . Έπειτα με τη βοήθεια ενός προγράμματος περιήγησης, αναζητώντας το http://localhost:8080 βλέπουμε την χρήσιμη διεπιφάνεια χρήστη του master που παρέχει το Spark για παρουσίαση σημαντικών πληροφοριών σχετικά με τα αναλυτικά στοιχεία των slaves που έχουν συνδεθεί στο cluster, συνολική μνήμη και πυρήνες που έχει το cluster, καθώς επίσης url του master μαζί με τη θύρα στην οποία θα συνδεθούν οι client-slaves αλλά και αναλυτικές πληροφρορίες για την εκτέλεση κάθε task και το standard output του κάθε executor.

Από τη μεριά των client τώρα, είναι αναγκαία η ύπαρξη του jdk 8, για αυτό το λόγο κατέβασα και εκτέλεσα την παρακάτω εντολή export JAVA_HOME ̄/home/alex/jdk1.8.0_91 και επαλήθευσα την ορθότητα με το echo $JAVA_HOME, έπειτα έγινε η εκκίνηση των workers και η σύνδεσή τους στο master με την εκτέλεση της εντολής ξανά μέσα από το directory home/alex/ που έχει γίνει η εγκατάσταση του Spark σε κάθε virtual machine .spark-1.6.1binhadoop2.6/sbin/start-slave.sh spark://192.168.1.24:7077, όπου το spark://192.168.1.24:7077 είναι το url που είναι σηκωμένος ο master. Η σύνδεση των workers στον master μπορεί να επαληθευτεί από την διεπιφάνεια χρήστη από τη μεριά του master ότι δηλαδή είναι alive οι συνδέσεις.

Επιπλέον, κάποιες σημαντικές παρατηρήσεις, είναι ότι και τα 3 μηχανήματα έχουν ίδιο όνομα υπολογιστή καθώς επίσης ο βασικός, ο πρώτος χρήστης που δημιουργήθηκε είχε επίσης το ίδιο όνομα και αυτό γιατί όταν μοιράζονται κοινό NFS directory οι clients θα υιοθετούν τις ρυθμίσεις για R/W access αυτού που 'μοιράζει' το directory, η έκδοση του Spark πρέπει να είναι η ίδια εκγατεστημένη σε όλα τα μηχανήματα, spark-1.6.1, το οποίο είναι σημαντικό, διότι αρχικά παρατηρήθηκε μη δυνατότητα σύνδεσης των slaves στο master όταν χρησιμοποιούνταν διαφορετικές εκδόσεις. Επίσης το JAVA_HOME path πρέπει να είναι είναι ρυθμισμένο σωστά και στα 3 μηχανήματα. Για την τελευταία παρατήρηση άνοιξα το αρχείο .bashrc με διακιώματα διαχειριστή με την εντολή sudo gedit .bashrc, πρόσθεσα στη τελευταία γραμμή το JAVA_HOME με το αντίστοιχο μονοπάτι στο οποίο είναι εγκατεστημένο το bin του java -version για κάθε μηχάνημα, αποθήκευσα τις αλλαγές στο αρχείο και έκανα επανεκκίνηση κάθε μηχάνημα για να αποθηκευτούν οι αλλαγές μόνιμα.

Επιπλέον κάποιες συμπληρωματικές πληροφορίες που αφορούν την εκτέλεση του παράλληλου προγράμματος με το Apache Spark, σε γλώσσα προγραμματισμού Java, είναι ότι χρειάστηκε η εγκατάσταση του maven. Εγκαταστάθηκε λοιπόν η έκδοση Apache Maven 3.3.9 και στα 3 μηχανήματα και πρόσθεσα στις μεταβλητές περιβάλλοντος του καθενός το bin της εγκατάστασης του maven, η εγκατάσταση έγινε κάτω από το home directory και στα 3 μηχανήματα ανοίγοντας το αρχείο .bashrc με δικαιώματα χρήστη και προσθέτοντας στη τελευταία γραμμή την ακόλουθη εντολή PATH = "$HOME/apache-maven-3.3.9/bin:$PATH" και έγινε και επανεκκίνηση σε κάθε μηχάνημα για να αποθηκευτούν οι αλλαγές.

Το πρόγραμμα έχει τη μορφή πακέτου maven δηλαδή στη ρίζα του αρχείου του προγράμματος βρίσκεται ο φάκελος src και το pom.xml με τα dependencies καθώς επίσης και ο παραγόμενος φάκελος target. Κάτω από το φάκελο src υπάρχει ο φάκελος main, κάτω από τον οποίο βρίσκεται ο φάκελος java ο οποίος περιέχει τις κλάσεις του προγράμματος. Πολλές φορές για γρήγορες και μικρές αλλαγές η επεξεργασία των κλάσεων γίνονταν κατευθείαν από το φάκελο java με έναν απλό editor αλλά η ανάπτυξη έγινε κυρίως στο Intellij λόγω της άνεσης που προσφέρει στην μετατροπή συγκεκριμένων τμημάτων κώδικα σε lambda expressions.

3. ΑΝΑΛΥΣΗ ΤΗΣ ΠΡΟΣΕΓΓΙΣΗΣ

3.1 Θεωρητική διατύπωση του προβλήματος

Το πρόβλημα που καλείται να εξεταστεί στη παρούσα πτυχιακή εργασία, είναι καταρχήν καταπόσο μπορεί να παραλληλοποιηθεί με τις δομές που υποστηρίζει το Apache Spark, ο ακολουθιακός αλγόριθμος που έχει μετατραπεί σε Java και αφετέρου κατά πόσο μπορεί να υπάρξει βελτιστοποιήση του συνολικού χρόνου εκτέλεσης μιας μεθόδου υλοποίησης του αλγορίθμου SVD, που έχει παρθεί από τη πηγή με σύνδεσμο http://www.timelydevelopment.com/demos/NetflixPrize.aspx. Ουσιαστικά θα γίνει σύγκριση για την ορθότητα των αποτελεσμάτων και του χρόνου διεκπεραίωσης της σειριακής υλοποίησης του αλγορίθμου με αυτήν της παράλληλης αφού προηγουμένως έχει προηγηθεί η κατάλληλη μετατροπή σε γλώσσα προγραμματισμού Java και του σειριακού προγράμματος αλλά και αφότου διαπιστωθεί ότι είναι εφικτή η παραλληλοποίηση του αλγορίθμου στη διεπιφάνεια προγραμματισμού Apache Spark και γίνει η εξήγηση και των δύο προγραμμάτων. Η απόπειρα για τη προσπάθεια βελτιστοποιήσης θα γίνει με παράλληλη υλοποίηση του αλγορίθμου σε περιβάλλον ανάπτυξης Apache Spark με γλώσσα προγραμματισμου Java και θα ακολουθήσει η σύγκριση με την σειριακή υλοποίηση του αλγορίθμου για να εξεταστεί πρωταρχικά αν ο συνολικός χρόνος εκτέλεσης του προγράμματος ελαττώνεται το οποίο είναι δευτερεύον ζητούμενο και έπειτα οι διαφορές στους χρόνους για κάθε function που καλείται από την εναρκτήρια main συνάρτηση μεταξύ των δύο υλοποιήσεων αλλά και το κύριο ζητούμενο το οποίο είναι κατά πόσο αλλάζει ή παραμένει ίδιος ο δείκτης μέσου όρου της απόλυτης τιμής της διαφοράς μεταξύ του rating και predict rating των αποτελεσμάτων, ο οποίος ιδανικά θα παρέμενε όμοιος.

3.2 Απαιτούμενη λειτουργικότητα

Η απαιτούμενη λειτουργκότητα και για το σειριακό πρόγραμμα αλλά και για το παράλληλο πρόγραμμα πρέπει να περιλαμβάνει αρχικά την δήλωση και αρχικοποίηση του μέγιστου αριθμού των βαθμολογιών, των πελατών, των ταινιών, των features και των εποχών καθώς επίσης και τον ελάχιστο αριθμό των εποχών για το dataset που θα χρησιμοποιηθεί, έπειτα την δημιουργία του constructor έτσι ώστε να συμπεριληφθούν και να αρχικοποιηθούν οποιαδήποτε στοιχεία και οποιεσδήποτε μεταβλητές και δομές δεδομένων χρειαστούν στην συνέχεια της υλοποίησης του κάθε προγράμματος, την ανάγνωση των αρχείων training file και testing file και την αποθηκευσή των πληροφοριών τους σε κατάλληλες δομές δεδομένων για την αντίστοιχη επεξεργασία που θα ακολουθηθεί σε κάθε περίπτωση και την υλοποίηση των μεθόδων LoadHistory, CalcFeatures, PredictRating και τελευταία τη ProcessTest για την αποθήκευση των αποτελεσμάτων των προβλέψεων στο predictions file και τελικά την καταγραφή των χρόνων για την υλοποίηση κάθε προαναφερμένου τμήματος.

Τα αποτελέσματα που αποθηκεύονται στο αρχείο predictions file από το παράλληλο πρόγραμμα σε Java πρέπει να παρουσιάζονται στην ίδια μορφή με τα αποτελέσματα που αποθηκεύονται στο αντίστοιχο predictions file στο σειριακό πρόγραμμα σε Java καθώς επίσης και στο σειριακό πρόγραμμα σε C++ που έχει παραμετροποιηθεί για να δέχεται τα dataset του MovieLens, δηλαδή 5 στήλες (χρήστης, ταινία, βαθμολογία, εκτιμώμενη βαθμολογία, απόκλιση εκτιμώμενης βαθμολογίας από αυτή που έχει δώσει ο χρήστης).

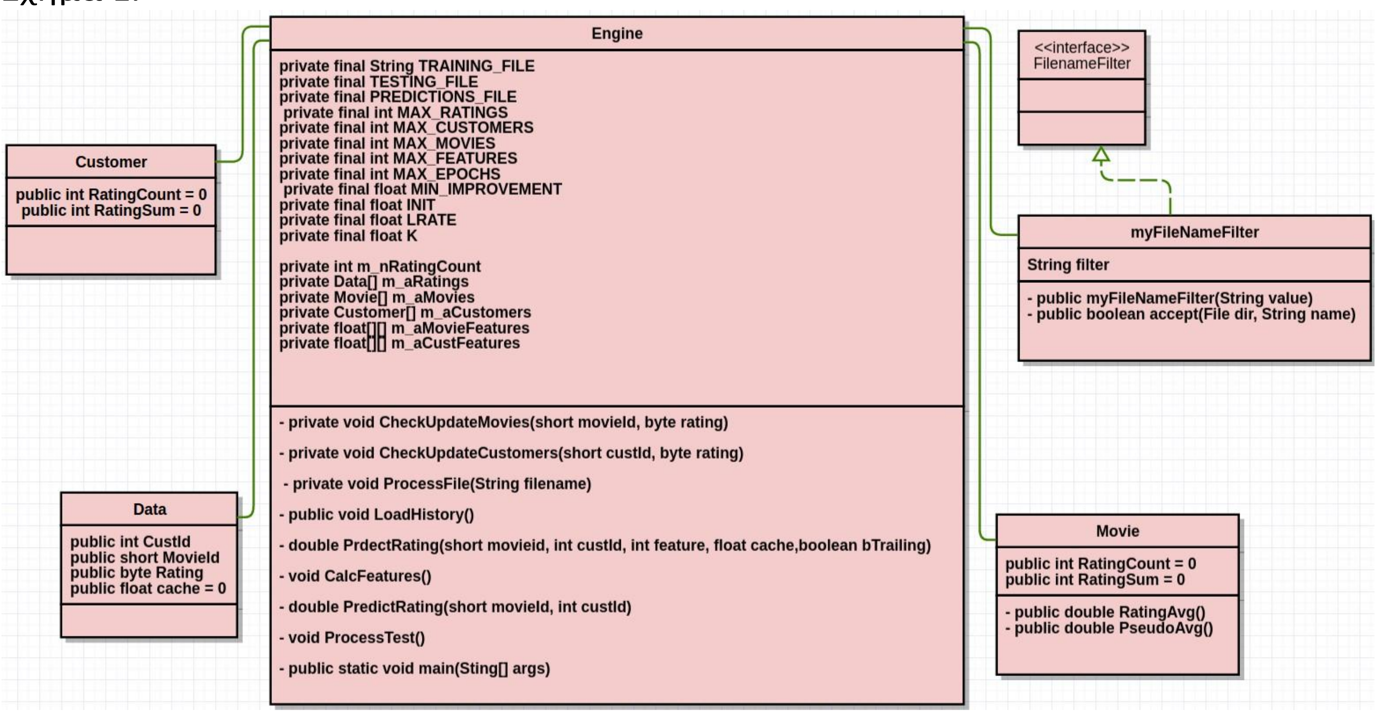
Τα αποτελέσματα από τα εξαγόμεα αρχεία predictions files που προκύπτουν από το σειριακό και παράλληλο πρόγραμμα πρέπει να είναι ακριβώς ίδια με τα αποτελέσματα από το prediction file που προκύπτει από την εκτέλεση του προγράμματος σε C++. ηλαδή όλες οι τιμές σε όλες τις στήλες να είναι ίδιες ακριβώς καθώς επίσης και ο αριθμός των συνολικών προβλέψεων σε κάθε αρχείο να είναι ίδιος εφόσον έχουν εκτελεστεί και τα 3 προγράμματα με MovieLens dataset ίδιου μεγέθους. Επίσης αφού υλοποιείται ίδιος αλγόριθμος πρέπει ο μέσος όρος απόκλισης της διαφοράς σε απόλυτη τιμή ανάμεσα στη βαθμολογία και εκτιμώμενη βαμολογία να είναι ο ίδιος σε όλα τα προγράμματα.

4. ΣΧΕΔΙΑΣΗ

4.1 Αρχιτεκτονικό διάγραμμα

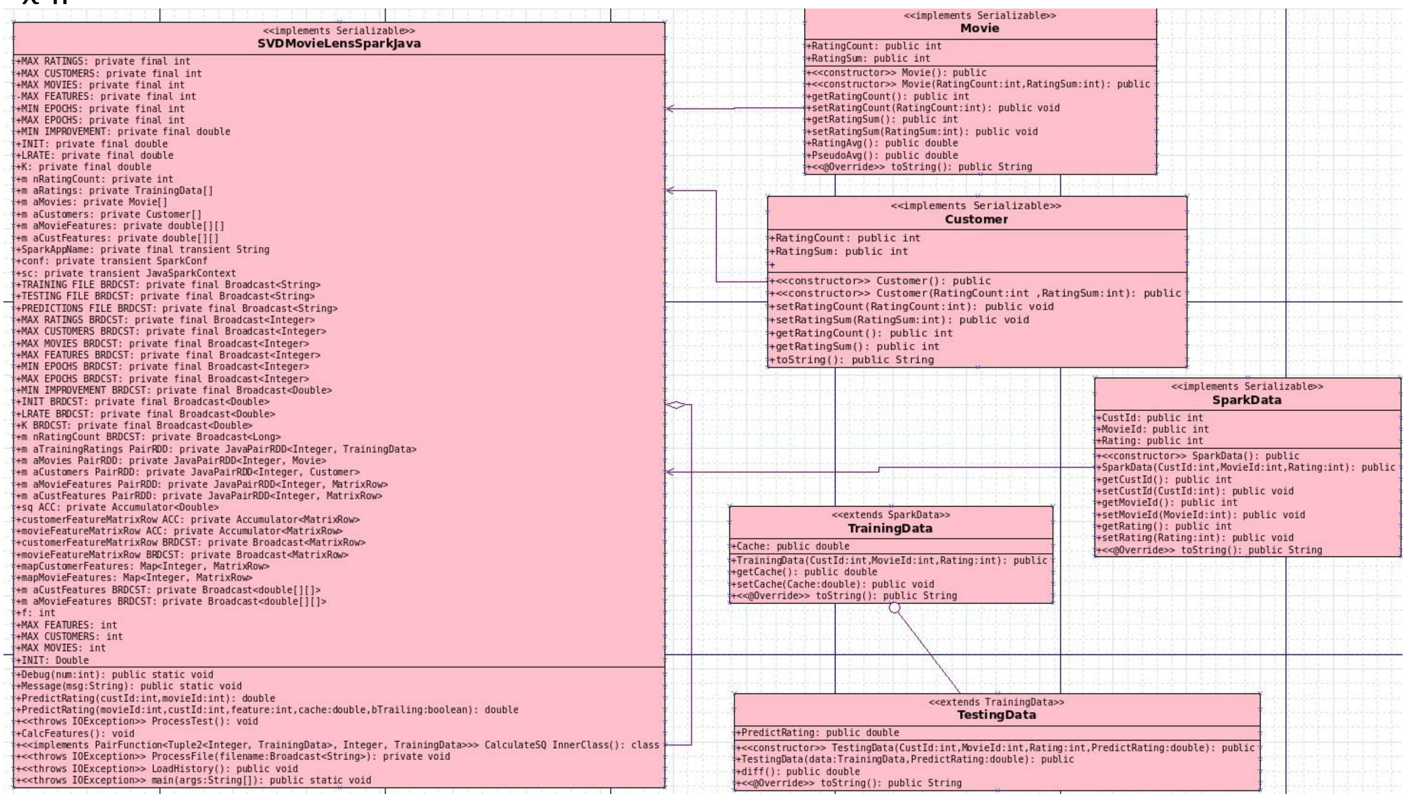
Ακολουθεί το διάγραμμα που αντικατοπτρίζει την υλοποίηση του σειριακού κώδικα σε Java.

Σχήμα 1.



Ακολουθεί το διάγραμμα που αντικατοπτρίζει την υλοποίηση του παράλληλου κώδικα σε Java.

Σχήμα 2.



4.2 Περιγραφή υποσυστημάτων - λειτουργιών

Περιγραφή ακολουθιακού αλγορίθ ου σε **C++ με MovieLens Dataset**

Αρχικά έπρεπε να γίνει τροποποιήση του αρχικού κώδικα που δόθηκε σε C++ έτσι ώστε να δέχεται ως είσοδο τα MovieLens Dataset. Καθώς τα dataset του MovieLens και του Netflix διαφέρουν ως προς το πλήθος και τη μορφή των training αρχείων εισόδου. Τα customer ID και movie ID στο NETFLIX δεν είναι «κανονικοποιημένα» σε περιοχές [1. . .] αλλά είναι οι κανονικοί κωδικοί πελάτη και ταινίας, με αποτέλεσμα να μην είναι δυνατή η άμεση χρήση τους σε διευθυνσιοδότηση πινάκων. Παρόλα αυτά έγινε η κατάλληλη μετατροπή στις κρίσιμες functions που διαβάζουν τα 2 αρχεία. Ο ίδιος που έχει γράψει τον αρχικό κώδικα (http://www.timelydevelopment.com/demos/NetflixPrize.aspx) αναφέρει ότι ο αλγόριθμος του μπορεί να χρησιμοποιηθεί και να τροποποιηθεί αρκεί να συμπεριληφθεί το μεγάλο σχόλιο που έχει προσθέσει ο ίδιος πάνω από το πρόγραμμα το οποίο περιέχει τις σχετικές ανακοινώσεις και αποδόσεις. Το πρόγραμμα το μετέτρεψα σε περιβάλλον Visual Studio 13 από τον χώρο εργασίας, καθώς μόνο εκεί είχα πρόσβαση στο εργαλείο αυτό.

Συνεπώς είναι επιτακτική η ανάγκη ύπαρξης αποτελεσμάτων από αυτό το πρόγραμμα έτσι ώστε να ανατρέχω σε αυτά για την επαλήθευση των αποτελεσμάτων που θα εξάγονται από τα 2 επόμενα προγράμματα που θα ακολουθήσουν, το σειριακό και το παράλληλο πρόγραμμα σε Java.

Ουσιαστικά ο αρχικός κώδικας ένα από τα σημεία στα οποία άλλαξε ήταν στα σημεία δήλωσης των μακροεντολών identifier. Αφαιρέθηκε το FEATURE_FILE και το TEST_PATH καθώς δεν χρειαζόντουσταν πλέον για το MovieLens. Τα TRAINING_PATH, TRAINING_FILE και PREDICTION_FILE διατηρήθηκαν με τα αντίστοιχα αρχεία του MovieLens. Όπως και τα υπόλοιπα αναγνωριστικά μακροεντολών διατηρήθηκανα και τους ανατέθηκαν οι αντίστοιχες τιμές του MovieLens dataset των εκατό χιλιάδων βαθμολογιών. Τα τρία structs που ακολουθούν τροποποιήθηκαν ελάχιστα. Στο struct Movie οι αρχικά δηλώμενες ως μεταβλητές RatingAvg και PseudoAvg αλλάχτηκαν ώστε να είναι δηλωμένες ως μέθοδοι που επιστρέφουν τον ίδιο τύπο δεδομένων με αυτό που είχαν δηλωθεί αρχικά. Στη struct Customer η μόνη αλλαγή που έγινε είναι ό,τι δεν συμπεριλήφθηκε η δήλωση του member του CustomerId. Και στο struct Data η μόνη μικροαλλαγή που έγινε είναι στο τύπο δήλωσης του member Cache από float να είναι doule.

Άλλες αλλαγές που έγιναν, είναι ότι πλέον η μέθοδος ProcessTest() δεν δέχεται καμία παράμετρο ενώ προηγουμένως δεχόταν σαν παράμετρο το όνομα του αρχείου που παραγόταν με τα αποτελέσματα.

Συνεχίζοντας στην κλάση Engine, στη δήλωση των δύο μονοδιάστατων πινάκων Movie και Customer η μόνη αλλαγή που έγινε είναι ότι δεσμεύτηκε χώρος για ένα στοιχείο παραπάνω. Στους δύο μονοδιάστατους πίνακες m_aMovieFeatures και m_aCustFeatures η μόνη αλλαγή ήταν να δεσμευτεί μια στήλη παραπάνω και να αλλάξει ο τύπος δεδομένων από float σε double. Το IdMap δεν χρειαζόταν για αυτό και δεν συμπερίφθηκε.

Στον constructor του Engine αυτό που αλλάζει είναι οι 2 βρόγχοι που κάνουν επαναλήψεις για τις στήλες των δύο δισδιάστατων πινάκων που προανέφερα καθώς η αρχικοποιήση πλέον ξεκινά από την 1 και μετά και φτάνει μέχρι MAX_MOVIES+1 και MAX_CUSTOMERS+1 αντίστοιχα. Αυτή η αλλαγή έγινε διότι στα MovieLens Dataset δεν υπάρχουν κωδικοί πελάτη και ταινίας με τιμή μηδέν, ξεκινούν από το 1, συνεπώς έπρεπε να γίνει η κατάλληλη τροποποιήση.

Επιπρόσθετα η CalcMetrics() δεν υπάρχει πλέον καθώς οι λειτουργίες τις υλοποιούνται πλέον στις άλλες κλάσεις και μεθόδους. Η LoadHistory() πλέον απλά καλεί τη μέθοδο ProcessFile(), παιρνώντας το

TRAINING_FILE ως παράμετρο, η οποία περιλαμβάνει και την υλοποιήση της παλιάς LoadHistory().

Στην νέα ProcessFile() η wchar_t pwzBuffer πλέον δεν χρησιμοποιείται. Επιπλέον δεν μας απασχολεί να διαβάσουμε την πρώτη γραμμή του αρχείου ξεχωριστά, καθώς το νέο dataset διαφέρει από αυτό του Netflix στο οποίο, κάθε αρχείο στην πρώτη σειρά είχε τον κωδικό της ταινίας που αντιπροσώπευε. Ενώ τώρα όλα η πληροφοριά είναι συγκεντρωμένη σε ένα αρχείο για όλες τους πελάτες και ταινίες. Συνεπώς στην νέα ProcessFile() με τον ίδιο τρόπο όπως γινόταν προηγουμένως με ένα pointer σε FILE και με την fopen ανοίγουμε την ροή για διάβασμα σε ένα αρχείο μέχρι να βρεθεί το τέλος του αρχείου. Μέσα στον επαναληπτικό βρόγχο με την βοήθεια της fscanf γίνεται το διάβασμα ενός set από τις τιμές των data χρησιμοποιώντας ως delimeter το tab για τις 4 στήλες που έχει το αρχείο και στο τέλος κάθε γραμμής χρησιμοποιώ τον ειδικό χαρακτήρα newline για να προχωρήσει στην επόμενη γραμμή. Στο τέλος κάθε επανάληψης, ο μετρητής m_nRatingCount αυξάνεται κατά ένα. Οι τιμές που διαβάζονται, αποθηκεύονται προσωρινά στις βοηθητικές μεταβλητές custId, movieId, rating και tmp. ΄Επειτα αναθέτονται το καθένα στα μέλη κάθε στοιχείου του m_aRatings με τη βοήθεια του φορέα πρόσβασης μελών και χρησιμοποιώντας ως index το m_aRatingCount. Επίσης πριν το τέλος του βρόγχου ενημερώνονται τα στατιστικά για το πλήθος και το άθροισμα των m_aMovies και των m_aCustomers.

Η μέθοδος CalcFeatures() παρέμεινε με ακριβώς την ίδια υλοποιήση. Οι δύο PredictRating() για τον γρήγορο υπολογισμό των trailing αλλά και τον υπολογισμό των τελικών αποτελεσμάτων επίσης παραμείνανε απαράλλαχτες.

Η ProcessTest() που έχει 2 βασικές λειτουργίες όμως, αλλάζει. Αρχικά πρέπει να φορτώσει ένα από τα αρχεία sample set που διαθέτει το MovieLens, αλλά αυτή τη φορά το format με το οποίο πρέπει να διαβαστούν είναι διαφορετικό. Επομένως ανοίγουμε 1 ροή για διάβασμα στο TESTING_FILE και 1 ροή για γράψιμο στο PREDICTION_FILE έτσι όπως γινόταν αρχικά αλλά πλέον το διάβασμα του αρχείου γίνεται έτσι όπως συνέβη στο TRAINING_FILE δηλαδή διάβασμα της οριοθετημένης γραμμής με tab, ύστερα αποθηκεύονται προσωρινά οι τιμές που διαβάστηκαν και αμέσως μετά καλείται η PredictRating() για τον τελικό υπολογισμό των features. Στη συνέχεια βρίσκουμε το diff δηλαδή την απόλυτη τιμή της διαφοράς μεταξύ βαθμολογίας και εκτιμώμενης βαθμολογίας για κάθε γραμμή, αθροίζουμε τις διαφορές αυτές και αυξάνουμε τον μετρητή σε κάθε επανάληψη. Πριν το τέλος του βρόγχου, γράφουμε σε κάθε γραμμή στην ίδια μορφή που διαβάσαμε τα άλλα δύο αρχεία, τα αποτελέσματα του custId, movieId, rating, predictrating και diff.

Μετά το πέρας του βρόγχου γράφουμε στη τελευταία γραμμή του αρχείου τα περιληπτικά στοιχεία, δηλαδή τον αριθμό των προβλέψεων και τον μέσο όρο της απόλυτης τιμής του diff.

Επιπλέον οι βοηθητικές μεταβλητές που χρησιμοποιούνται στον αρχικό κώδικα δεν χρειάζονται πλέον καθώς δεν υπάρχει ανάγκη για κάποιου είδους parse.

Περιγραφή ακολουθιακού αλγορίθμου σε **Java με MovieLens Dataset**

Έπειτα ακολούθησε η μετατροπή του ακολουθιακού κώδικα C++ που δέχεται ως είσοδο Movielens dataset σε ακολουθιακό κώδικα σε Java με MovieLens Dataset επίσης.

Το σειριακό πρόγραμμα, αποτελείται από 5 κλάσεις συνολικά. Αρχικά υπάρχει η κλάση του Customer στην οποία απλά ορίζονται δύο πεδία και αρχικοποιούντε με 0. Τα δύο αυτά πεδία είναι το RatingCount το οποίο κρατάει τον αριθμό των συνολικών ψηφοφοριών που έχει πάρει μέρος ο πελάτης και το RatingSum που θα αποθηκεύει το συνολικό άθροισμα των βαθμολογιών που έχει δώσει ο πελάτης.

Η δεύτερη κλάση, είναι η Data, στην οποία δηλώνονται τέσσερα πεδία, το πρώτο πεδίο είναι το CustId το οποίο είναι η ταυτότητα του πελάτη, το δεύτερο πεδίο είναι το MovieId το οποίο είναι η ταυτότητα της ταινίας, το τρίτο είναι το Rating το οποίο αποτελεί τη βαθμολογία που έχει δώσει ο πελάτης στη ταινία και η Cache η οποία είναι αρχικοποιημένη με 0.

Τρίτη κλάση, είναι το Movie στο οποίο δηλώνονται και αρχικοποιούνται με 0 δύο πεδία το RatingCount το οποίο κρατά το πλήθος των ψηφοφοριών για μια ταινία και το RatingSum το οποίο κρατά το άθροισμα των βαθμολογιών που έχει πάρει μια ταινία. Επίσης στη κλάση δηλώνονται 2 μέθοδοι το RatingAvg() που υπολογίζει το μέσο όρο των βαθμολογιών και το PseudoAng() το οποίο υπολογίζει το ψευδο-μέσο όρο. Και τα 2 προηγουμένως υπολογίζονται χρησιμοποιώντας το RatingCount και το RatingSum αλλά κάνοντας διαφορετική πράξη.

Τέταρτη κλάση, είναι το myFileNameFilter το οποίο υλοποιεί ένα interface που ονομάζεται FilenameFilter και χρησιμοποιείται για το φιλτράρισμα λιστών καταλόγων. Στη κλάση αυτή δηλώνεται μια μεταβλητή filter και υλοποιούνται ο constructor του myFileNameFilter και η μέθοδος accept η οποία παίρνει σαν παραμέτρους το μονοπάτι για το αρχείο και το όνομά του έτσι ώστε να φιλτραριστεί το αρχείο και να επιστρέψει η μέθοδος true ή false.

Πέμπτη κλάση, είναι η Engine στην οποία δηλώνονται ολόκληρα τα μονοπάτια για το u.data το οποίο αποτελεί το training file με όλες τις βαθμολογίες, u1.test το οποίο αποτελεί το testing fie και το u1.predictions το οποίο αποτελεί το predictins file όπου γράφονται τα αποτελέσματα. Έπειτα ακολουθεί η δήλωση του μέγιστου αριθμού βαθμολογιών, πελατών, ταινιών, φεατυρες και εποχών καθώς επίσης και ο ελάχιστος αριθμός εποχών.

Στη συνέχεια, ορίζονται ο ελάχιστος δείκτης βελτίωσης του feature, η αρχική τιμή των features, η παράμετρος ποσοστιαίας εκμάθησης και η παράμετρος κανονικοποιήσης που χρησιμοποείται για την ελαχιστοποιήση της περιγραφής του τυχαίου σφάλματος ή θορύβου. Μετά ακολουθεί η δήλωση ενός counter ο οποίος αναλαμβάνει να αποθηκεύει τον τρέχων αριθμό των φορτωμένων βαθμολογιών, ένας μονοδιάστατος πίνακας με τα δεδομένα των βαθμολογιών, ένας άλλος μονοδιάστατος πίνακας με τις μετρικές των ταινιών (πλήθος και άθροισμα) και ένας ακόμα μονοδιάστατος πίνακας με τις μετρικές των πελατών.

Τελικά, δηλώθηκαν δύο δυσδιάστατοι πίνακες, ο πρώτος αποθηκεύει τα features με βάση τις ταινίες και ο δεύτερος αποθηκεύει τα feautures με βάση τους πελάτες. Μετά τη δήλωση των πεδίων, ακολουθεί η δήλωση του constructor στον οποίο γίνται η αρχικοποιήση του counter με τον τρέχων αριθμό φορτωμένων βαθμολογιών και με εμφωλευμένους επαναληπτικούς βρόγχους γίνεται η αρχικοποιήση των 2 προηγούμενων δυσδιάστατων πινάκων.

Ύστερα, ακολουθεί η δήλωση των μεθόδων, πρώτη μέθοδος είναι η CheckUpdateMovies() η οποία παίρνει σαν είσοδο δύο παραμέτρους, τη ταυτότητα της ταινίας και τη βαθμολογία της, εξετάζει να δει αν μια ταυτότητα ταινίας υπάρχει ήδη και ανανεώνει τις μετρικές της ταινίας αυτής, αν δεν υπάρχει τότε πρώτα δημιουργεί μια ταινία με τη καινούρια ταυτότητα και ακολούθως αυξάνει τις μετρικές της. εύτερη μέθοδος είναι η CheckUpdateCustomers() η οποία επίσης παίρνει 2 παραμέτρους, τη ταυτότητα του πελάτη και τη βαθμολογία του και εξετάζεται αν υπάρχει ήδη αυτή η ταυτότητα του πελάτη στο μονοδιάστατο πίνακα m_aCustomers[] και ανανεώνονται οι μετρικές του. ιαφορετικά αν δεν υπάρχει δημιουργείται νέος πελάτης με αυτή την ταυτότητα και αυξάνονται οι μετρικές του.

Μετέπειτα, ακολουθεί η μέθοδος ProcessFile() η οποία παίρνει μια παράμετρο, το μονοπάτι του αρχείου training file, και γίνεται ανάγνωση του αρχείου με τη βοήθεια του BufferedReader και Scanner, των 3 πεδίων που μας ενδιαφέρουν, της ταυτότητας του πελάτη, της ταινίας και της βαθμολογίας ανά γραμμή χρησιμοποιώντας ως delimeter το tab. Στη συνέχεια γίνεται ανανέωση των μετρικών του πίνακα με τις βαθμολογίες με βάση τις τιμές που διαβάσαμε από το αρχείο για μια γραμμή. Έπειτα καλούνται οι μέθοδοι CheckUpdateMovies() και CheckUpdateCustomers() για την ανανέωση των δικών τους στατιστικών. Τέλος αυξάνεται ο counter με τον τωρινό αριθμό φορτωμένων βαθμολογιών. Η διαδικασία που περιγράφτηκε υλοποιείται κάθε φορά που διαβάζεται νέα γραμμή από τον inputStream ο οποίος αποτελεί ένα BufferedReader. Σε περίπτωση που η προηγούμενη διαδικασία δεν πετύχει, τυπώνεται ένα μήνυμα λάθους στο χρήστη που τον ενημερώνει για την αποτυχία στο άνοιγμα του αρχείου. Τελικά ανεξάρτητα αν πετύχει η διαδικασία ή όχι μπαίνουμε στο μπλοκ Finally όπου σταματάμε τη ροή του BufferedReader και του Scanner.

Επόμενη μέθοδος, είναι η LoadHistory(), η οποία απλά καλεί τη προηγούμενη ProcessFile() παιρνώντας της ως παράμετρο, το μονοπάτι για το αρχείο training file.

΄Επειτα υλοποιείται η μέθοδος PredictRating() στην οποία περνιούνται πέντε παράμετροι ως είσοδος, τη ταυτότητα της ταινίας και του πελάτη, το feature, τη cache και το bTrailing το οποίο είναι προαιρετικό για τον υπολογισμό νέας τιμής για τη cache. Στην αρχή ελέγχεται η cache, αν έχουμε παλιά features τότε παίρνουμε την cached τιμή αλλιώς επιλέγεται το 1 ως η προτεινόμενη τιμή για το μέσο. ΄Επειτα υπολογίζεται το γινόμενο του feature ως προς τη ταυτότητα της ταινίας με το feature ως προς τη ταυτότητα του πελάτη και τελικά προστίθεται στη μεταβλητή sum, ουσιαστικά προστίθεται η συμμετοχή του τωρινού feature. Στις ακραίες περιπτώσεις όπου η βαθμολογία είναι πάνω από 5 τότε στο sum ανατίθεται η τιμή 5 και στη περίπτωση που η βαθμολογία είναι κάτω από 1 τότε ανατίθεται η τιμή 1 στο sum. Τελικά η PredictRating() επιστρέφει το sum.

Επόμενη μέθοδος, είναι η CalcFeatures() στην οποία δηλώνονται η ταυτότητα του πελάτη και της ταινίας και αρχικοποιείται η τελευταία όπως και ένας counter με 0 μαζί με κάποιες άλλες βοηθητικές μεταβλητές για τους επαναληπτικούς βρόγχους. ΄Επειτα, δηλώνεται ένα αντικείμενο τύπου Data το rating, η μεταβλητή δείκτη σφάλματος και μέσης τετραγωνικής ρίζας σφάλματος. Στη συνέχεια, αρχικοποιείται η μεταβλητή της ρίζας του μέσου τετραγωνικού σφάλματος με 2 και η τελευταία μεταβλητή ρίζας του τετραγωνικού σφάλματος με 0. Στη συνέχεια σκανάρονται όλα τα Features και αναθέτονται τα πεδία του rating σε αντίστοιχες τοπικές μεταβλητές της μεθόδου έτσι ώστε να υπολογιστεί στη συνέχεια η βαθμολογία και ο δείκτης σφάλματος για τον ελάχιστο αριθμό των εποχών ή μέχρι ωσότου να σταματήσει να γίνεται σημαντική πρόοδος, δηλαδή η διαφορά του δείκτη μέσης τετραγωνικής ρίζας σφάλματος με τον ελάχιστο δείκτη βελτίωσης να είναι μικρότερος του δείκτη ρίζας μέσου τετραγωνικού σφάλματος. Επίσης στην προηγούμενη επαναληπτική διαδικασία γίνονται cache off οι προηγούμενες τιμές αποθηκεύοντας σε δύο βοηθητικές μεταβλητές τις cf και mf των features καθώς θα χρειαστούν στην συνέχεια των εφμωλευμένων επαναληπτικών βρόγχων για τον υπολογισμό των νέων τιμών των features με βάση τους πελάτες και τους τις ταινίες. Πριν το τέλος του δεύτερου εμφωλευμένου βρόγχου, γίνεται ο υπολογισμός της ρίζας του μέσου τετραγωνικού σφάλματος. Μπορεί να τυπωθεί προαιρετικά αυτή η πληροφορία για αυτό έχει προστεθεί σε σχόλια η εντολή εκτύπωσης. Τελικά, συνεχίζοντας στο τελευταίο κομμάτι του δεύτερου εμφωλευμένου βρόγχου ακολουθεί ένας άλλος cache off οι παλιές προβλέψεις για τις βαθμολογίες καλώντας επαναληπτικά τη μέθοδο PredictRating() παιρνώντας τις πέντε παραμέτρους που χρειάζεται, για τη ταυτότητα ταινίας και πελάτη, τον δείκτη του feature τη Cache του αντικειμένου rating και τη τελευταία παράμετρο btrailing να είναι false.

Ακολουθεί πάλι η μέθοδος PredictRating() αλλά σε αυτή τη περίπτωση έχει ως είσοδο μόνο δύο παραμέτρους τη ταυτότητα της ταινίας και του πελάτη, στην οποία γίνεται αρχικοποιήση μια μεταβλητής sum με το 1. Έπειτα ακολουθεί ένας επαναληπτικός βρόγχος ο οποίος διατρέχει όλα τα features των δυσδιάστατων πινάκων m_aMovieFeatures[][] και m_aCustFeatures[][] για τη συγκεκριμένη ταινία και πελάτη, κρατώντας σταθερό και στους δύο πίνακες τους δείκτες των στηλών δεδομένου των παραμέτρων που έχουν περαστεί με τη κλήση της μεθόδου. Υπολογίζεται το γινόμενό τους και πριστίθεται στη μεταβλητή sum. Στις ακραίες περιπτώσεις που το άθροισμα της βαθμολογίας είναι μεγαλύτερο του 5 αντισταθμίζεται αναθέτοντάς του την τιμή 5 και στην περίπτωση που είναι μικρότερο του ενός του εκχωρείται η τιμή 1. Τέλος επιστρέφεται η τιμή του sum από τη μέθοδο.

Ύστερα, ακολουθεί η μέθοδος ProcessTest() στην οποία γίνεται ανάγνωση του testing file, το οποίο όπως αναφέρθηκε και προηγουμένως αποτελεί ένα δείγμα του συνόλου με το ίδιο format που έχει το u.data, το οποίο έχει όλο το σύνολο των βαθμολογιών και επίσης γράφονται τα αποτελέσματα στο prediction file. Αρχικά, αρχικοποιούνται τρία αντικείμενα τύπου BufferedReader, PrintWriter και ένας Scanner, με τη βοήθεια των οποίων θα γίνει η ανάγνωση και το γράψιμο των 2 προηγούμενων αρχείων. Επίσης, αρχικοποιείται η ταυτότητα του πελάτη και της ταινίας, μαζί με την βαθμολογία, την προβλεπόμενη βαθμολογία που θα προκύψει από τις αριθμητικές πράξεις σύμφωνα με την υλοποίηση του κώδικα που έχει υλοποιηθεί σε C++ για το διαγωνισμό του Netflix. Συνεχίζοντας, γίνεται δήλωση της μεταβλητής diff, η οποία θα κρατάει την απόλυτη διαφορά μεταξύ της βαθμολογίας και της προτεινόμενης βαθμολογίας, ένας αθροιστής ο οποίος αρχικοποιείται με 0 όπως επίσης και ένας counter. Γίνεται δήλωση ενός αντικειμένου data τύπου Data, και ξεκινάει μια ροή για την ανάγνωσή του με τη βοήθεια του FileReader και του BufferedReader. Αμέσως μετά, ξανά με τη βοήθεια των δύο προηγούμενων αντικειμένων ανοίγει το αρχείο testing file αυτή τη φορά μαζί με τη μέθοδο PrintWriter, με τη βοήθεια της οποίας θα γίνει το γράψιμο των προβλέψεων στο predictions file. Με τη μέθοδο radLine() του αντικειμένου inputStream, που είναι τύπου BufferedReader γίνεται η ανάγνωση κάθε γραμμής των αρχείων, πρώτα γίνεται η ανάγνωση της ταυτότητας του πελάτη και της ταινίας και η βαθμολογία από το αρχείο testing file και έπειτα καλείται η μέθοδος PredictRating() για τον υπολογισμό της πρόβλεψης της βαθμολογίας παιρνώντας ως παραμέτρους μόνο τη ταυτότητα της ταινίας και του πελάτη. Μετά, υπολογίζεται η διαφορά της αρχικής βαθμολογίας με αυτήν της προβλεπόμενης σε απόλυτη τιμή και προστίθεται στον αθροιστή και αυξάνεται ο counter κατά ένα για κάθε γραμμή. Τελικά, γράφονται οι προβλέψεις στο αρχείο predictions file τοποθετώντας τη ταυτότητα του πελάτη, έπειτα της ταινίας και τη βαθμολογία και μετά την υπολογισμένη προβλεπόμενη βαθμολογία και από δίπλα την απόκλιση που έχει από την αρχική βαθμολογία, χρησιμοποιώντας το tab ως delimeter.

Στη περίπτωση που αποτύχει η παραπάνω διαδικασία, γίνεται catch το σφάλμα για αποτυχία ανοίγματος κάποιου αρχείου και ενημερώνεται ο χρήστης κατάλληλα. Τελικά, στο μπλοκ του finally που θα εκτελεστεί σε οποιαδήποτε περίπτωση γίνεται το κλείσιμο ροής του αρχείου που έχει ανοίξει ο inputStream για το testing file, καθώς επίσης το κλείσιμο του out που χρησιμοποιήθηκε για το γράψιμο των αποτελεσμάτων στο predictions file, καθώς επίσης και του βοηθητικού scanner. Τελικά τυπώνεται στη κονσόλα ένα μύνημα με τον υπολογισμό της μέσης εκτίμησης σφάλματος για τα προηγούμενα αποτελέσματα.

Τελευταία είναι η μέθοδος main() από όπου γίνεται η εκκίνηση του προγράμματος. Η main() ξεκινά με τη δήλωση 5 μεταβλητών που θα χρησιμοποιηθούν για την αποθήκευση των χρονικών στιγμών πριν και μετά από τη δημιουργία ενός αντικειμένου τύπου Engine και τη κλήση κάθε μεθόδου έτσι ώστε να υπολογιστούν οι χρονικές διαφορές στους και να τυπώσει το πρόγραμμα στη κονσόλα τη χρονική διάρκεια για την εκτέλεση κάθε βήματος ξεχωριστά. Πρώτα ακολουθεί η δημιουργία του αντικειμένου engine τυπώνεται ο χρόνος για τη δημιουργία του αντικειμένου, μετά ακολουθεί η κλήση της μεθόδου LoadHistory() του αντικειμένου engine και έπειτα τυπώνεται η χρονική διάρκεια εκτέλεσής της σε δευτερόλεπτα. Μετά, γίνεται κλήση της μεθόδου CalcFeatures() του engine και αντίστοιχα εκτυπώνεται η χρονική διάρκεια για την εκτέλεση της μεθόδου. Τελευταία μέθοδος που καλείται είναι η ProcessTest() του engine και τελικά τυπώνεται η ο χρόνος εκτέλεσής της καθώς επίσης και ένα ειδοποιητήριο μήνυμα Done για το τερματισμό του σειριακού προγράμματος.

5. ΥΛΟΠΟΙΗΣΗ

5.1 Λεπτομέρειες υλοποίησης

Περιγραφή παράλληλου αλγορίθμου σε **Java** στο **Apache Spark** με **MovieLens Dataset**

Πριν την υλοποιήση του παράλληλου προγράμματος ακολούθησε μια διαδικασία εξοικείωσης με το Apache Spark. Διάβασα τα σχετικά Documentation από το site του Spark, όπως Spark Overview, Quick Start, Spark Programming Guide, API Docs (Java), Spark Examples (Java), Cluster Mode Overview, Spark Standalone Mode κτλπ. Μετά ακολούθησε εξοικίωση με το Scala Shell για την εκτέλεση αλληλεπιδραστικών παραδειγμάτων. Επιπλέον διάβασα αρκετές διαφορές ανάμεσα στις δυνατότητες της Scala και της Java. Για παράδειγμα η parallelize του sparkContext της Scala κάνει απευθείας μετατροπή μήτρες, ενώ η αντίστοιχη μέθοδος του javaSparkContext απαιτεί μονοδιάστατο πίνακα (vector). Κατανόηση της παραλληλοποίησης μέσω Spark (RDD, Broadcast variables και Accumulator variables και tranformations/actions σε RDDs). Επιλογή κατάλληλων δομών δεδομένων σε Spark Java στις οποίες θα μετατραπούν οι δομές δεδομένων του ακολουθιακού κώδικα Java.

Συγκεκριμένα ο πίνακας Ratings μετατράπηκε σε Broadcast RDD variable για βελτιστοποίηση της επικοινωνίας. ιάφορες σταθερές (static final) μεταβλητές μετατράπηκαν σε Broadcast variable αντίστοιχου τύπου, επίσης για βελτιστοποίηση επικοινωνίας.

Δεν υπάρχει παραλληλοποιήσιμη R/W δομή που να υποστηρίζει δισδιάστατους πίνακες. Αυτό οφείλεται στην immutable(αμετάβλητη) φύση των RDD που είναι και η βασική παραλληλοποιήσιμη δομή στο Apache Spark. Έγιναν απόπειρες υλοποίησης της παραλληλοποίησης με δικής μου έμπνευσης κλάσεις. Αυτές οι απόπειρες έχουν συμπεριληφθεί στον κώδικα αλλά δεν καλούνται από την main() και επομένως δεν εκτελούνται.

Τα σημεία του κώδικα-αλγορίθμου στα οποία εντόπισα δυνατότητα να επιτευχθεί απευθείας παραλληλοποίηση ήταν τρία, τα οποία είναι τα εξής:

1. Το διάβασμα του αρχείου δεδομένων.

2. Ο υπολογισμός των prediction ratings για το test file με υπολογισμένους ήδη τους πίνακες customer και movie features.

3. Το γράψιμο του αρχείου αποτελεσμάτων.

Αρχικά λοιπόν το παράλληλο πρόγραμμα αποτελείται από τις εξής κλάσεις, Customer, Movie, SparkData, TrainingData, TestingData και SVDMovieLensSparkJava. Η εκκίνηση του προγράμματος γίνεται από την main() η οποιά εμπεριέχεται στην SVDMovieLensSparkJava. ΄Ολες οι κλάσεις κάνουν implement το Serializable έτσι ώστε να εκτεθούν δημόσια οι λεπτομέρειες όλων των κλάσεων. Χρησιμοποιεί ένα universal identifier για κάθε Serializable class ο οποίος κατά τη διάρκεια του deserialization βοηθά στο να εξασφαλιστεί ότι η φορτωμένη κλάση αντιστοιχεί ακριβώς σε ένα serialized object. Αν δεν βρεθεί αντιστοιχία τότε ρίχνει InvalidClassException.

Main()

```
public static void main(String[] args) throws IOException {
LocalTime t1, t2, t3, t4,    t5;

t1  = LocalTime.now();
SVDMovieLensSparkJava engine = new SVDMovieLensSparkJava();
t2   =  LocalTime.now();
System.out.printf("engine construction duration equals %d s\n",Duration.between(t1,t2).getSeconds());
engine.LoadHistory();
t3  = LocalTime.now();
System.out.printf("load history duration equals %d s\n",Duration.between(t2,t3).getSeconds());
engine.CalcFeatures();
t4  = LocalTime.now();
System.out.printf("calculation feature duration equals %d s\n",Duration.between(t3,t4).getSeconds());
engine.ProcessTest();
t5  = LocalTime.now();
System.out.printf("processing test duration equals %d s\n",Duration.between(t4,t5).getSeconds());
System.out.println("\nDone\n");
}
```

Συνεχίζοντας, στην κλάση του Customer δηλώνονται και αρχικοποιούνται με μηδέν δύο data members το RatingCount και το RatingSum. Υπάρχει ο default constructor που απαιτείται για το Serialization και ο δεύτερος constructor που αναθέτει τις τιμές που παίρνει σαν παραμέτρους στο RatingCount και το RatingSum αντίστοιχα. Έπειτα ακολουθούν οι setters και getters για τα μέλη. Τελικά δηλώνεταιη μέθοδος toString() που αναλαμβάνει να τυπώσει τα προηγούμενα πεδία με τις τιμές τους.

Έπειτα ακολουθεί η κλάση Movie στην οποία επίσης δηλώνονται και αρχικοποιούνται με μηδέν τα δύο δικά της data members, RatingCount και RatingSum. Στην συνέχεια δημιουργούνται οι δύο ςονστρυςτορ καθώς επίσης οι setters και οι getters των μελών. Μετά ακοουθούν η μέθοδος RatingAvg() που υπολογίζει τον μέσο όρο των βαθμολογιών που έχει πάρει μια ταινία και την PseudoAvg() που υπολογίζει αντίστοιχα τον ψεύδο-μέσο όρο μιας ταινίας. Τελικά η toString() τυπώνει τα πεδία και τις αντίστοιχες τιμές τους.

Έπειτα ακολουθεί η κλάση SparkData, στην οποία δηλώνονται τρία data members το CustId, το MovieId και το Rating. Μετά ακολουθούν ο default constructor και ο constructor που τις αναθέτει τιμές των παραμέτρων που παίρνει στα πεδία της κλάσης. Ύστερα ακολουθούν οι setters και οι getters των πεδίων της κλάσης. Στο τέλος υπάρχει η αντίστοιχη tostring() μέθοδος που τυπώνει τα πεδία και τις τιμές των CustId, MovieId και Rating.

Μετέπειτα ακολουθεί η TrainingData κλάση, η οποία κληρονομεί την κλάση SparkData με όλα τα πεδία και μεθόδους της. Επιπλέον δηλώνεται και αρχικοποιείται με μηδέν το πεδίο Cache. Έπειτα ο constructor της TrainingData καλεί τον πατρικό constructor χρησιμοποιώντας την super. Ακολουθούν ο setter και τεξτλατινγεττερ για την Cache. Τελικά υπάρχει η μέθοδος toString() που τυπώνει τα πεδία της πατρικής κλάσης μαζί με την Cache.

Στη συνέχεια έρχεται η TestingData που κληρονομεί με την σειρά της την TrainingData κλάση. η-λώνεται ως data member το PredictRating και στην συνέχεια ακολουθούν οι constructor της κλάσης που με την σειρά τους καλούν με την βοήθεια της super τον πατρικό τους constructor. Ο πρώτος constructor παίρνει σαν παράμετρο το CustId, MovieId, Rating και PredictRating και ο δεύτερος παίρνει δύο παραμέτρους το TrainingData και το PredictRating. Μετά ακολουθεί η μέθοδος diff() που υπολογίζει και επιστρέφει την απόλυτη τιμή της διαφοράς μεταξύ βαθμολογίας και εκτιμώμενης βαθμολογίας. Τελικά ακολουθεί η toString() της TestingData που με την σειρά της θα χρησιμοποιηθεί για να γραφτούν τα αποτελέσματα στο αρχείο αποτελεσμάτων.

Τελευταία κλάση είναι η SVDMovieLensSparkJava, αρχικά δηλώνονται ο μέγιστος αριθμός βαθμολογιών, πελατών και ταινιών σε ολόκληρο το training set. Έπειτα δηλώνονται ο μέγιστος αριθμός των feature που είναι να χρησιμοποιηθούν καθώς επίσης και ο μέγιστος και ελάχιστος αριθμός των εποχών ανά feature.

Δηλώνονται επίσης και ο δείκτς ελάχιστης βελτίωσης που απαιτείται για να συνεχιστεί το τρέχον feature και η τιμή αρχικοποιήσης για τα features καθώς επίσης η παράμετρος του ρυθμού εκμάθησης και η παράμετρος κανονικοποιήσης που χρησιμοποιείται για την ελαχιστοποιήση της υπερπροσαρμογής. Επιπλέον δηλώθηκε το πεδίο που θα μετράει τον τωρινό αριθμό των φορτωμένων βαθμολογιών. Επίσης δηλώθηκαν οι τρεις μονοδιάστατοι πίνακες που κρατάνε τις πληροφορίες των βαθμολογιών, τις μετρήσεις για τις ταινίες και τις μετρήσεις για τους πελάτες αλλά και οι δισδιάστατοι πίνακες που κρατάνε τα features ανά ταινία και ανά πελάτη.

Τα πεδία που δηλώθηκαν μέχρι στιγμής υπήρχαν και στη σειριακή εκδοχή αλλά χρησιμοποιούνται και στο παράλληλο πρόγραμμα για την εκτέλεση της μεθόδου CalcFeatures(), τώρα ακολουθούν τα πεδία που χρησιμοποιούνται από τις μεθόδους που εκτελούνται παράλληλα. Αρχικά δηλώνεται και ανατίθεται το όνομα της εφαρμογής του Spark προγράμματος, μετά δηλώνεται η σύνδεση spark context στο Spark cluster που θα χρησιμοποιηθεί στην συνέχεια για την δημιουργία RDDs, accumulators και broadcast variables σε εκείνο τον cluster και το spark conf που χρησμιμοποιείται από το προηγούμενο το οποίο επιτρέπει την ρύθμιση ορισμένων κοινών ιδιοτήτων. Στη συνέχεια δηλώνονται τρεις broadcast variables το TRAINING_FILE_BRDCST, το TESTING_FILE_BRDCST και το PREDICTIONS_FILE_BRDCST έτσι ώστε κάθε κόμβος να έχει ένα αντίγραφο του input dataset με έναν αποδοτικό τρόπο. Με τον ίδιο τρόπο δηλώνονται και όλα τα προηγούνενα πεδία που αφορούν το σειριακό κομμάτι με εξαίρεση βέβαια τους 3 μονοδιάστατους και τους δύο δισδιάστατους πίνακες. Οι πίνακες πλέον έχουν δηλωθεί ως JavaPairRDDs έτσι ώστε να φέρουμε τα δεδομένα σε μορφή key-value RDDs που θα δώσουν στην συνέχεια την δυνατότητα εκτέλεσης συναθροίσεων, ομαδοποίηση δεδομένων με το ίδιο κλειδί και ομαδοποιήση δύο διαφορετικών RDDs. Και στους πέντε πίνακες στη θέση του key παίρνουν Integer generic data type, ενώ στη θέση του value οι τρεις μονοδιάστατοι παίρνουν αντικείμενα τύπου TrainingData, Movie και Customer ενώ οι δύο δισδιάστατοι στη θέση του αλυε έχουν δηλωθεί έτσι ώστε να δέχονται MatrixRow δηλαδή ένα row oriented κατανεμημένο πίνακα χωρίς να έχουν κάποια σημασία οι δείκτες σειράς. Έπειτα ακολούθησε η δήλωση τριώ accummualator έτσι ώστε με ασφάλεια να ενημερώνονται οι μεταβλητές για τον υπολογισμό του sq και για τον υπολογισμό των Featureture των πελατών και των ταινιών όταν η εκτέλεσή τους θα γίνεται ξεχωριστά σε όλους τους worker κόμβους στο cluster . Δηλώθηκαν πεδία επίσης για τον υπλογισμό των Featureture των πελατών και των ταινιών αλλά με τη διαφορά τώρα ότι είναι δηλωμένες ως Broadcast. Μετά ακολουθεί η δήλωση δύο Map του mapCustomerFeatures και του mapMovieFeatures. Και τελικά δηλώνονται δύο δισδιάστατοι πίνακες broadcast που αποθηκεύουν τα feature ανά Movie και Customer.

Στην συνέχεια φτιάχτηκαν δύο μέθοδοι η Debug και η Message οι οποίες αποτέλεσαν τα δύο βασικά μου εργαλεία για την αποσφαλμάτωση του κώδικα κατά την διάρκεια της ανάπτυξης, αφού το ίδιο το Apache Spark δεν υποστηρίζει κάποιο debugging σύστημα.

Debugging Methods

```
public static void Debug(int num) {
        System.out.printf("|----------------------> DBG %d\n", num);
    }

    public static void Message(String msg) {
        System.out.printf("|-----> %s <----|\n", msg);
    }
```

Μετέπειτα, ακολουθεί ο constructor της SVDMovieLensSparkJava μέσα στον οποίο γίνεται δημιουργία του Spark Configuration Object. Είχε γίνει μια απόπειρα να χρησιμοποιηθεί μιας δικιάς μου custom έκδοσης Kryo Registrator για βελτιστοποιήση στον χρόνο η οποία τελικά λειτουργούσε αλλά μόνο όσο έκανα απόπειρες για παραλληλοποιήση της μεθόδου της CalcFeatures() η οποία και τελικά απέτυχε για τους λόγους που έχω καταγράψει αναλυτικά στην ενότητα 7.2 στην οποία μάλιστα παρουσιάζω τμήματα κώδικα συμπεριλαμβανομένου του Kryo Registrator και των άλλων προσπαθειών, συμπερασματικά χρησιμοποιήθηκε ο default registrator που παρέχεται από το Spark. Στην συνέχεια γίνεται αρχικοποιήση του spark context συμπεριλαμβανομένου και των υπόλοιπων πεδίων που δηλώθηκαν στο τμήμα του Spark Data Members.

Ύστερα γίνονται cache οι τιμές των broadcast μεταβλητών σε τοπικές μεταβλητές έτσι ώστε να επιτευχθεί βελτιστοποιήση στον χρόνο. Στην συνέχεια δημιουργείται ένα RDD για τον πίνακα με τα customer feature χρησιμοποιώντας αυτοδύναμη παραγωγή δεικτών. Ακολουθεί ο επαναληπτικός βρόγχος που διατρέχει όλα τα features για να δημιουργηθεί ένα vector από όλους τους πελάτες με την broadcast τιμή αρχικοποιήσης. Στη συνέχεια το vector αυτό με τη βοήθεια της parallelize βοηθά στη δημιουργία μιας παράλληλης συλλογής RDD από μια υπάρχουσα συναλλαγή.

Σε αυτή τη παράλληλη συλλογή που δημιουργήθηκε υλοποιούμε το transformation zipWithIndex το οποίο προσφέρει ένα σταθερό indexing, απαριθμώντας κάθε στοιχείο κατά την αρχική σειρά του. Επειδή όμως οι δείκτες με αυτό το transformation εμφανίζονται δεξιά από τις τιμές, κάνουμε επιπρόσθετα ένα ακόμα transformation το mapToPair το οποίο επιστρέφει ένα νέο RDD εφαρμόζοντας μια function σε όλα τα στοιχεία του RDD. Σε αυτή τη περίπτωση η fuction αυτό που υλοποιεί είναι αντιστροφή των θέσεων των δύο στοιχείων έτσι ώστε ο index να βρίσκεται από μπροστά. Αυτό επιτυγχάνεται με τη βοήθεια του Tuple2 το οποίο επιτρέπει το ζευγάρωμα δύο αντικειμένων αλλά και των μεθόδων που υποστηρίζει αυτό για να γίνεται πρόσβαση στο πρώτο και δεύτερο αντικείμενο του Tuple2 αλλά και στα επιμέρους πεδία των αντικειμένων αυτών. Τέτοιου είδους προσβάσεις σε στοιχεία Tuple2 γίνονται πολύ συχνά στην συνέχεια.

Έπειτα ακολουθεί η δημιουργία των RDDs για τον πίνακα με τα Movie Feature, χρησιμοποιώντας αυτοδύναμη παραγωγή δεικτών. Γίνεται αντίστοιχα η δημιουργία του vector με όλες τις ταινίες με την broadcast τιμή αρχικοποιήσης. Το vector μετατρέπεται σε RDD στο οποίο αντίστοιχα εφαρμόζεται η zipWithIndex και η mapToPair έτσι να επιτευχθεί το indexing που επιθυμούμε έτσι όπως έγινε με το RDD με τους πελάτες προηγουμένως.

Δημιουργία RDD's για features των ταινιών και πελατών

```
for (f = 0; f < MAX_FEATURES; f++) {
// create a vector of MAX_CUSTOMERS with INIT_BRDCST value
matrixRowList.add(new MatrixRow(Collections.nCopies(MAX_CUSTOMERS, INIT)));
}
JavaRDD<MatrixRow> rdd1 = sc.parallelize(matrixRowList);
this.m_aCustFeatures_PairRDD = rdd1.zipWithIndex().
mapToPair((Tuple2<MatrixRow, Long> tuple) -> new Tuple2<Integer, MatrixRow>(Integer.valueOf(tuple.
_2.intValue()), tuple._1));
// Create RDDs for Movie Feature matrix, using autogenerated indices
matrixRowList = new ArrayList<>();
for (f = 0; f < MAX_FEATURES; f++) {
   // create a vector of MAX_MOVIES with INIT_BRDCST value
matrixRowList.add(new MatrixRow(Collections.nCopies(MAX_MOVIES, INIT)));
}
this.m_aMovieFeatures_PairRDD = sc.parallelize(matrixRowList).zipWithIndex().
mapToPair((Tuple2<MatrixRow, Long> tuple) -> new Tuple2<Integer, MatrixRow>(Integer.
valueOf(tuple._2.intValue()), tuple._1));
```

Τα transformations πάνω στα RDDs έγιναν με lambda expression.

Μετέπειτα έγινε η κατασκευή των maps από τα RDDs με τα Customer και Movie Features.

Δημιουργία HashMaps από τα RDD's με τα feature των ταινιών και πελατών

```
// create maps from the Customer and Movie Feature RDDs
        mapCustomerFeatures = new HashMap<>(m_aCustFeatures_PairRDD.collectAsMap());
        mapMovieFeatures = new HashMap<>(m_aMovieFeatures_PairRDD.collectAsMap());
```

Αμέσως μετά γίνεται η αρχικοποιήση των δύο δισδιάστατων πινάκων m_aMovieFeatures και m_aCustFeatures έτσι όπως γινόταν προηγουμένως και στο σειριακό πρόγραμμα.

Μετέπειτα έρχεται η σειρά της main() η οποία ουσιαστικά δημιουργεί ένα αντικείμενο τύπου SVDMovieLensSparkJava, έπειτα καλείται η LoadHistory, μετά η CalcFeatures και τελικά η ProcessTest. Όλες οι μέθοδοι που θα εξηγήσω στην συνέχεια υλοποιήθηκαν παράλληλα εκτός από την CalcFeatures() η οποία εκτελείται ακολουθιακά και παρέμενει ως έχει. Καταγράφονται οι χρονικές στιγμές πριν και μετά από την εκτέλεση κάθε μεθόδου έτσι ώστε να μετρηθούν και να καταγραφούν οι χρόνοι εκτέλεσης έτσι ακριβώς όπως έγινε και στο σειριακό πρόγραμμα.

Η πρώτη μέθοδος είναι η LoadHistory() που κάνει throw για I/O exception και απλά καλεί την μέθοδο ProcessFile() παιρνώντας ως παράμετρο το TRAINING_FILE_BRDCST.

Έπειτα στην ProcessFile που και αυτή κάνει throw για I/O exception γίνεται δημιουργία ενός αρχικού RDD από String από το διάβασμα του training data αρχείου. Στην συνέχεια γίνεται ο υπολογισμός του αριθμού των βαθμολογιών και τελικά δημιουργείται ένα RDD για τα στατιστικά του Customer. Για τη δημιουργία αυτού του RDD χρησιμοποιώντας τη map πάνω στο trainingFile διαβάζεται το αρχείο με το tab ως delimeter και το αποτέλεσμα ανατίθεται στο columnsTrainingFile.

Διάβασμα training αρχείου

```
JavaRDD<String>    trainingFile    =  this.sc.textFile(filename.value());
        // calculate the number of  ratings
        this.m_nRatingCount_BRDCST  =  this.sc.broadcast(trainingFile.count());
```

Στην συνέχεια στο columnsTrainingFile γίνεται το transformation mapToPair έτσι ώστε να επιστραφεί ένα RDD με τους κωδικούς των πελατών και τις βαθμολογίες του και θα ανατεθεί στο columnsCustomersA_ το οποίο είναι δηλωμένο ως PairRDD από Integer και Integer. Μετά στο columnsCustomersA_ γίνεται transformation reduceByKey έτσι ώστε να βρεθεί το άθροισμα των βαθμολογιών που έχει δώσει ο κάθε πελάτης συνολικά, τα αποτελέσματα ανατίθονται στο columnsCustomersA το οποίο αποτελεί ένα PairRDD. Έπειτα ξανά γίνεται transformation στο columnsTrainingFile έτσι ώστε να πάρουμε αυτή τη φορά τους πελάτες και για κάθε πελάτη ανατίθεται στο πεδίο του value ο αριθμός ένα και τελικά ανατίθενται στο PairRDD columnsCustomersB_. Ύστερα στο columnsCustomersB_ γίνεται το transformation reduceByKey έτσι ώστε να βρεθεί το πλήθος των βαθμολογιών που έχει δώσει κάθε πελάτης τα οποία και ανατίθενται στο columnsCustomersB. Στη συνέχεια γίνεται join των δύο PairRDDs columnsCustomersA και columnsCustomersB έτσι ώστε να επιστραφεί ένα PairRDD με το key να είναι Integer που θα είναι ο κωδικός πελάτη και value να είναι ένα Tuple2 με key να είναι Integer το οποίο θα είναι το πλήθος των βαθμολογιών που έχει δώσει κάθε πελάτης και value να είναι επίσης Integer με το άθροισμα των βαθμολογιών που έχει δώσει ο κάθε πελάτης. Το PairRDD αυτό ανατίθεται στο columnsCustomers. Και τελικά το columnsCustomers που προκύπτει αντιστοιχίζεται στο .m_aCustomers_PairRDD το οποίο είναι δηλωμένο ως PairRDD με Integer και Customer object. Συνεπώς στο πεδίο του Integer αντιστοιχείται ο κωδικός πελάτη και στο Customer object του οποίου η κλάση έχει δηλωθεί αντιστοιχίζονται τα πεδία RatingCount και RatingSum τα οποία και υπάρχουν στο columnsCustomers.

Δημιουργία RDD για τα στατιστικά των πελατών

```
// create  RDD  for  Customer statistics
JavaRDD<String[]> columnsTrainingFile = trainingFile.map(line -> line.split("\t"));
JavaPairRDD<Integer, Integer> columnsCustomersA_
= columnsTrainingFile.mapToPair((String[] row) -> new Tuple2(Integer.
parseInt(row0]),          Integer.parseInt(row[2])));
JavaPairRDD<Integer, Integer> columnsCustomersA = columnsCustomersA_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer,Integer>  columnsCustomersB_  = columnsTrainingFile.
```

```
mapToPair((String[] row) -> new Tuple2(Integer.parseInt(row[0]), 1));
JavaPairRDD<Integer,Integer> columnsCustomersB = columnsCustomersB_.
reduceByKey((a, b) -> a +    b);
JavaPairRDD<Integer, Tuple2<Integer, Integer>> columnsCustomers = 0columnsCustomersB.
join(columnsCustomersA);
this.m_aCustomers_PairRDD = columnsCustomers.mapToPair((Tuple2<Integer,
Tuple2<Integer,  Integer>>  tuple)  ->  new  Tuple2<Integer,    Customer>(tuple.
_1,  new  Customer(tuple._2._1,  tuple._2._2)));
```

Με αντίστοιχη διαδικασία γίνεται και η δημιουργία του RDD για τα στατιστικά των ταινιών. Εντοπίζεται το πλήθος και το άθροισμα των βαθμολογιών για κάθε ταινία και τελικά φορτώνται οι κωδικοί των ταινιών, τα πλήθη και τα αθροίσματα των βαθμολογιών ανά ταινία στο m_aMovies_PairRDD. Μετέπειτα ακολουθεί η δημιουργία του πίνακα με τις βαθμολογίες από τα Data αντικείμενα, πάνω στα RDD columnsTrainingFile, rddData και rddIndexedData γίνονται τα transformations map, zipWithIndex και mapToPair αντίστοιχα και προκύπτει το τελικό m_aTrainingRatings_PairRDD .

Δημιουργία RDD για τα στατιστικά των ταινιών

```
 // create RDD for Movie  statistics
JavaPairRDD<Integer, Integer> columnsMoviesA_= columnsTrainingFile.
mapToPair((String[]  row)  ->  new  Tuple2(Integer.parseInt(row[1]),   Integer
.parseInt(row[2])));
JavaPairRDD<Integer, Integer> columnsMoviesA = columnsMoviesA_.
reduceByKey((a,  b)  ->  a  +  b);
JavaPairRDD<Integer, Integer> columnsMoviesB_ = columnsTrainingFile.
mapToPair((String[]  row)  ->  new  Tuple2(Integer.parseInt(row[1]),   1));
JavaPairRDD<Integer, Integer> columnsMoviesB = columnsMoviesB_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer, Tuple2<Integer, Integer>> columnsMovies = columnsMoviesB.
join(columnsMoviesA);
this.m_aMovies_PairRDD  =  columnsMovies.mapToPair((Tuple2<Integer,  Tuple2<Integer,  Integer>>  tuple)  ->
new  Tuple2<Integer,  Movie>(tuple._1,  new  Movie(tuple._2._1,  tuple._2._2)));
```

```
// create Ratings array with Data  objects
JavaRDD<TrainingData>  rddData  =  columnsTrainingFile.map((String[]  row)
 -> new TrainingData(Integer.parseInt(row[0]), Integer.parseInt(row[1]), Integer.parseInt(row[2])));
JavaPairRDD<TrainingData, Long> rddIndexedData = rddData.zipWithIndex();
this.m_aTrainingRatings_PairRDD = rddIndexedData.
mapToPair((Tuple2<TrainingData, Long> tuple) -> new Tuple2(tuple._2.intValue(), tuple._1));
```

Στην συνέχεια γίνεται ενημέρωση των μελών της σειριακής έκδοσης

```
this.m_nRatingCount = this.m_nRatingCount_BRDCST.getValue().intValue();
this.m_aCustomers_PairRDD.collectAsMap().forEach((k,v)->this.m_aCustomers[k]=v);
this.m_aMovies_PairRDD.collectAsMap().forEach((k, v) -> this.m_aMovies[k] = v);
this.m_aTrainingRatings_PairRDD.collectAsMap().forEach((k,v)->this.m_aRatings[k]=v);
```

Έπειτα έρχεται η σειρά της CalcFeatures() η οποία δεν άλλαξε και παραμένει ακολουθιακή, διότι δεν γίνεται να παραλληλοποιηθεί και για αυτό εκτελείται μόνο από τον driver. Αυτό ουσιαστικά δεν είναι δύσκολο να επιτευχθεί αρκεί να αναλογιστεί κανείς τα εξής ακόλουθα, οτιδήποτε πραγματοποιείται στο εσωτερικό του closure από transformations όπως (map, filter, groupBy, aggregateBy, κτλπ.) υλοποιείται σε executor ή σε executors. Στα προηγούμενα συμπεριλαμβάνονται η ανάγνωση δεδομένων από απομακρυσμένες πηγές ή persistent storage. Ενώ actions όπως το count, reduce, κτλ. συνήθως εκτελούνται και από τον driver και τους executors, σε αυτές τις περιπτώσεις ο μεγαλύτερος φόρτος εργασίας αναλαμβάνεται να υλοποιηθεί παράλληλα και μερικά τελευταία βήματα εκτελούνται διαδοχικά στον driver. Όλα τα υπόλοιπα που κάνουν trigger ένα action ή transformation συμβαίνουν στον driver. Πιο συγκεκριμένα κάθε ενέργεια που χρειάζεται πρόσβαση στο SparkContext εκτελείται στον driver.

Κατόπιν ακολουθεί η αρχικοποιήση των broadcast πινάκων m_aCustFeatures_BRDCST και m_aMovieFeatures_BRDCST έτσι ώστε να είναι διαθέσιμοι στους executors.

Αρχικοποίηση broadcast πινάκων

```
// initialize broadcast matrices to make them available in executors
        m_aCustFeatures_BRDCST = sc.broadcast(m_aCustFeatures);
        m_aMovieFeatures_BRDCST  = sc.broadcast(m_aMovieFeatures);
```

Μετέπειτα έρχεται η σειρά της τελευταίας μεθόδου ProcessTes() της οποίας οι βασικές λειτουργίες είναι το παράλληλο διάβασμα του test file και η εξαγωγή του αρχείου με τα αποτελέσματα predictions file παράλληλα. Αρχικά δηλώνεται ένας μετρητής ο οποίος κρατά τον αριθμό των βαθμολογιών στο test file και ένας αθροιστής sum ο οποίος αθροίζει τις απόλυτες τιμές των διαφορών μεταξύ βαθμολογιών και εκτιμώμενων βαθμολογιών. Αργότερα υλοποιείται το διάβασμα του test file με την textFile χρησιμοποιώντας ως delimeter το tab κρατάω από κάθε γραμμή μόνο τις πληροφορίες από τις πρώτες τρεις στήλες, η χρονοσφραγίδα δεν χρησιμοποιείται και όλα αυτά φορτώνονται στο RDD c_testingData_RDD.

ProcessTest()

```
    // Reads in parallel the test file and produces the predictions file in parallel
    void ProcessTest() throws IOException {
        long cnt; // number of test  ratings
        double  sum; // sum of Abs difference between Rating and PredictionRating
        // read test file and create RDD of TrainingData items
        JavaRDD<TrainingData>   c_testingData_RDD
                 = this.sc.textFile(this.TESTING_FILE_BRDCST.getValue()).
        map(line    ->    line.split("\t")).
        map((String[]   row)   ->   new TrainingData(Integer.
        parseInt(row[0]),  Integer.parseInt(row[1]),  Integer.parseInt(row[2])));
        // transform RDD of TrainingData to RDD of TestingData items with our predictions
        JavaRDD<TestingData>   testingDataRDD
                 =   c_testingData_RDD.
                 map((TrainingData  t)  ->  new  TestingData(t,  PredictRating(t.CustId,  t.MovieId)));
        // count test ratings
```

```
        cnt  =  testingDataRDD.count();
        // sum the abs differences between ratings and predictionratings
        sum = testingDataRDD.map(t -> t.diff()).reduce((a, b) -> a + b);
        // save RDD of TestingData to prediction file
        testingDataRDD.saveAsTextFile(this.PREDICTIONS_FILE_BRDCST.getValue());
        System.out.printf("\n--\nNumber of predictions: %d\nAvg Abs(diff): %f\n",  cnt,sum/cnt);
    }
```

Μετά ακολουθεί η μετατροπή του RDD με τα TrainingData σε RDD το testingDataRDD με τα TestingData μαζί με τις προβλέψεις, καλώντας την PredictRating() και παιρνώντας της ως παραμέτρους τον κωδικό πελάτη και ταινίας. Το μόνο που αλλάζει στην υλοποιήση της PredictRating() είναι η χρήση των δύο broadcast δισδιάστατων πινάκων για να γίνεται ο υπολογισμός παράλληλα από όλους τους executors, στη θέση των δύο παραδοσιακών δισδιάστατων πινάκων έτσι όπως υλοποιούνταν στη σειριακή εκδοχή. Τελικά μετριέται το πλήθος των βαθμολογιών στο testingDataRDD, γίνεται η άθροιση με την sum μεταβλητή και γίνεται η αποθήκευση του RDD στο prediction file με την saveAsTextFile. Για την ολοκήρωση του προγράμματος τυπώνονται ο συνολικός αριθμός των προβλέψεων που εξάγονται στο αρχείο και ο δείκτης μέσου όρου της απόλυτης τίμης της διαφοροποιήσης μεταξύ βαθμολογιών και εκτιμώμενων βαθμολογιών το οποίο προκύπτει από τη διαίρεση του sum με το cnt.

PredictRating(int,int)

```
    // Used by RDD transformation
double PredictRating(int custId, int movieId) {
double sum = 1;
int f =  0;

for (f = 0; f < this.MAX_FEATURES_BRDCST.getValue(); f++) {
 sum += this.m_aMovieFeatures_BRDCST.getValue()[f][movieId] * this.
 m_aCustFeatures_BRDCST.getValue()[f][custId];
  if (sum > 5) {
      sum = 5;
```

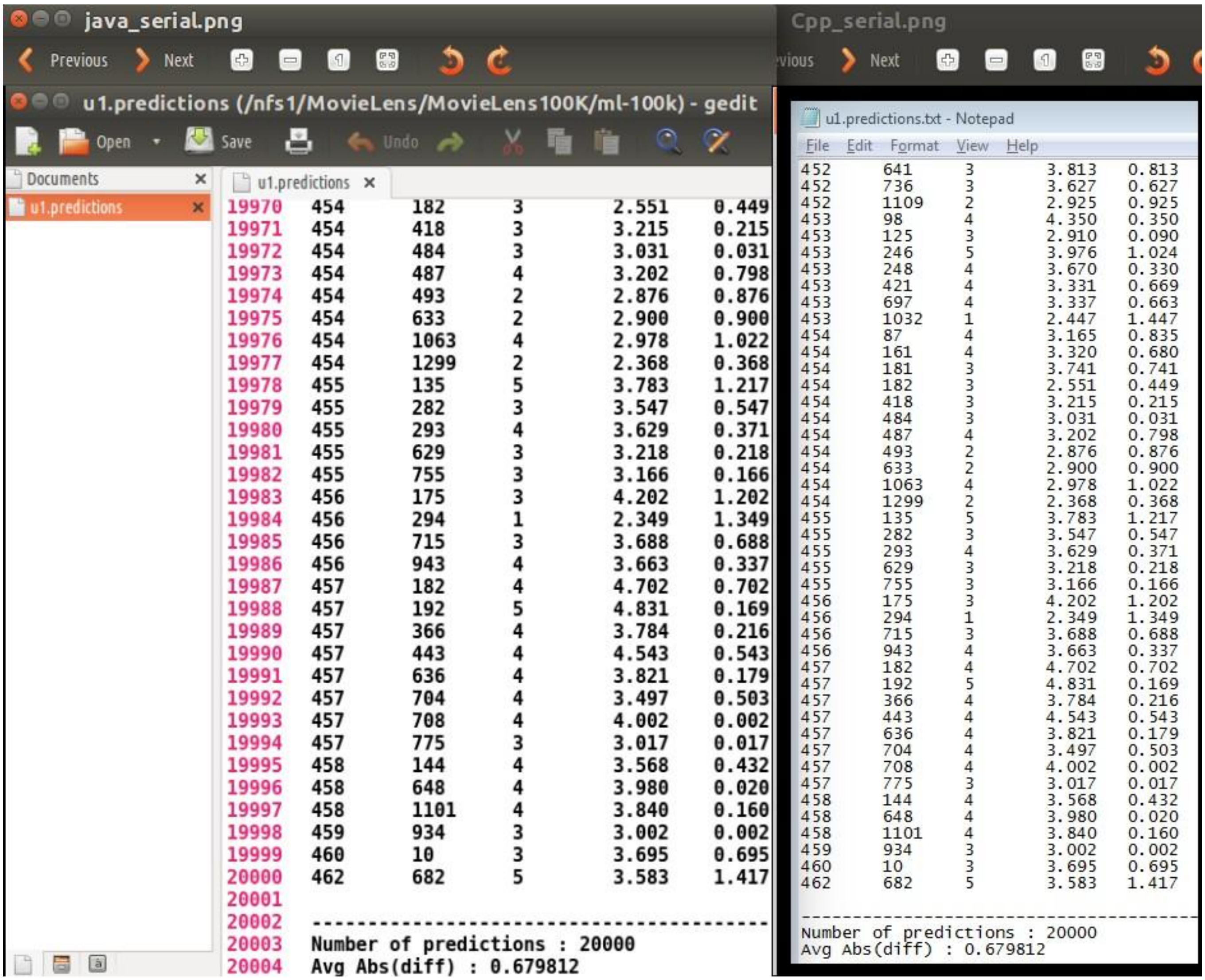
```
    }
    if (sum < 1) { sum = 1;
    }
}
    return sum;
}
}
```

5.2 Οθόνες εφαρμμογής

Ο λόγος που κρίθηκε αναγκαία η εκτέλεση του αρχικού προγράμματος σε C++ ήταν έτσι ώστε να διαπιστωθεί καλύτερα η λειτουργία του προγράμματος και έτσι ώστε να υπάρχει μια σαφέστερη εικόνα των εξαγόμενων πληροφοριών του προγράμματος που στη συνέχεια χρησιμοποιήθηκε ως πυξίδα για την αντίστοιχη υλοποιήση σε Java.

Ακολουθεί η εικόνα από τα αρχεία με τα αποτελέσματα που έχουν εξαχθεί έτσι ώστε να μπορώ να καθοδηγούμαι από αυτά και να μπορώ να τα συγκρίνω και να επαληθεύσω τα αποτελέσματα που προκύπτουν στη συνέχεια από το σειριακό και το παράλληλο πρόγραμμα σε Java. Επιπλέον τα αποτελέσματα ουσιαστικά που εξάγονται σε 5 στήλες με tab αναμεσά τους, είναι οι προηγούμενες πληροφορίες ταυτότητα πελατών και ταινιών, βαθμολογίες που έχουν ανατεθεί για τις ταινίες, εκτιμώμενες βαθμολογίες και τέλος στη τελευταία στήλη απόκλιση εκτιμώμενης βαθμολογίας από την αρχική βαθμολογία.

Εικόνα 1



19970	454	182	3	2.551	0.449
19971	454	418	3	3.215	0.215
19972	454	484	3	3.031	0.031
19973	454	487	4	3.202	0.798
19974	454	493	2	2.876	0.876
19975	454	633	2	2.900	0.900
19976	454	1063	4	2.978	1.022
19977	454	1299	2	2.368	0.368
19978	455	135	5	3.783	1.217
19979	455	282	3	3.547	0.547
19980	455	293	4	3.629	0.371
19981	455	629	3	3.218	0.218
19982	455	755	3	3.166	0.166
19983	456	175	3	4.202	1.202
19984	456	294	1	2.349	1.349
19985	456	715	3	3.688	0.688
19986	456	943	4	3.663	0.337
19987	457	182	4	4.702	0.702
19988	457	192	5	4.831	0.169
19989	457	366	4	3.784	0.216
19990	457	443	4	4.543	0.543
19991	457	636	4	3.821	0.179
19992	457	704	4	3.497	0.503
19993	457	708	4	4.002	0.002
19994	457	775	3	3.017	0.017
19995	458	144	4	3.568	0.432
19996	458	648	4	3.980	0.020
19997	458	1101	4	3.840	0.160
19998	459	934	3	3.002	0.002
19999	460	10	3	3.695	0.695
20000	462	682	5	3.583	1.417

20001
20002 --
20003 Number of predictions : 20000
20004 Avg Abs(diff) : 0.679812

452	641	3	3.813	0.813
452	736	3	3.627	0.627
452	1109	2	2.925	0.925
453	98	4	4.350	0.350
453	125	3	2.910	0.090
453	246	5	3.976	1.024
453	248	4	3.670	0.330
453	421	4	3.331	0.669
453	697	4	3.337	0.663
453	1032	1	2.447	1.447
454	87	4	3.165	0.835
454	161	4	3.320	0.680
454	181	3	3.741	0.741
454	182	3	2.551	0.449
454	418	3	3.215	0.215
454	484	3	3.031	0.031
454	487	4	3.202	0.798
454	493	2	2.876	0.876
454	633	2	2.900	0.900
454	1063	4	2.978	1.022
454	1299	2	2.368	0.368
455	135	5	3.783	1.217
455	282	3	3.547	0.547
455	293	4	3.629	0.371
455	629	3	3.218	0.218
455	755	3	3.166	0.166
456	175	3	4.202	1.202
456	294	1	2.349	1.349
456	715	3	3.688	0.688
456	943	4	3.663	0.337
457	182	4	4.702	0.702
457	192	5	4.831	0.169
457	366	4	3.784	0.216
457	443	4	4.543	0.543
457	636	4	3.821	0.179
457	704	4	3.497	0.503
457	708	4	4.002	0.002
457	775	3	3.017	0.017
458	144	4	3.568	0.432
458	648	4	3.980	0.020
458	1101	4	3.840	0.160
459	934	3	3.002	0.002
460	10	3	3.695	0.695
462	682	5	3.583	1.417

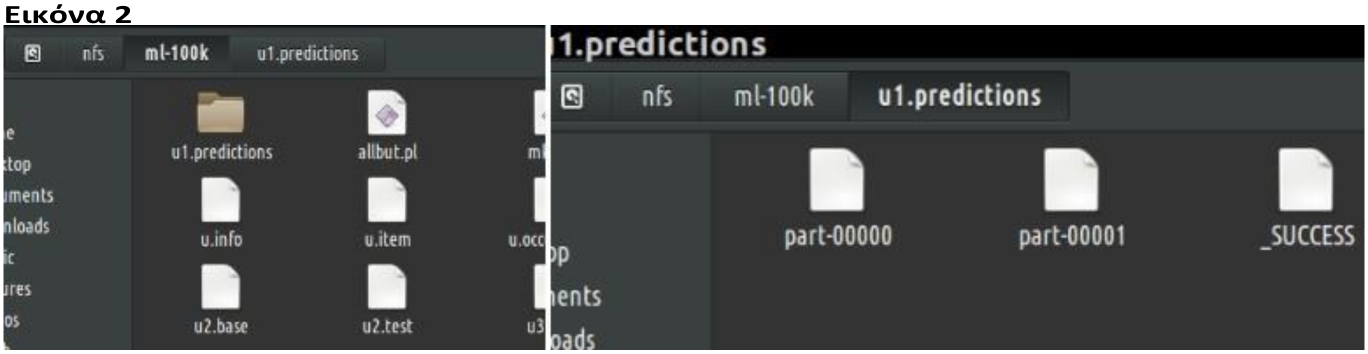
Number of predictions : 20000
Avg Abs(diff) : 0.679812

Από πάνω φαίνονται μαζί τα αποτελέσματα από τα predictions files που δημιουργήθηκαν από το σειριακό πρόγραμμα σε Java (αριστερά) και από το σειριακό πρόγραμμα σε C++ (δεξιά) έτσι ώστε να επαληθευτούν τα αποτελέσματα. Παρατηρούμε ότι τα αποτελέσματα και στις 5 στήλες (customer id, movie id, rating, predicted rating, abs(rating - predicted rating)) είναι ίδια. Καθώς επίσης και ο συνολικός αριθμός των προβλέψεων όπως επίσης και ο μέσος όρος της απόλυτης διαφοράς του rating - predicted rating είναι ακριβώς ίδια.

Ακολουθούν οι οθόνες εφαρμογής από το παράλληλο πρόγραμμα υλοποιημένο σε Java στο Apache Spark με το dataset των 100.000 βαθμολογιών.

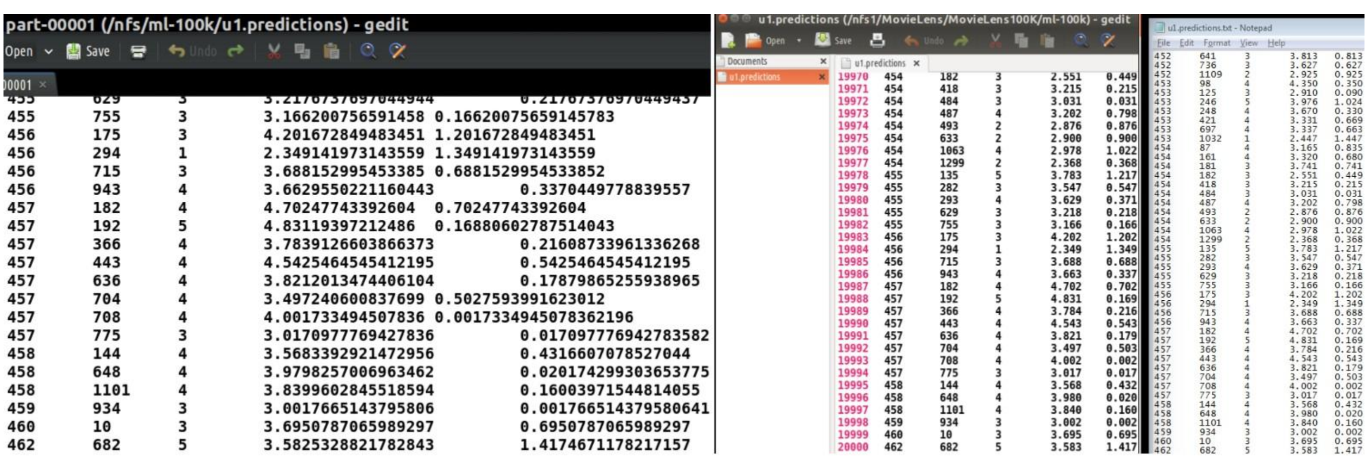
Από κάτω ακολουθούν εικόνες από το output στο directory του nfs. Όπως μπορούμε να διαπιστώσουμε η saveAsTextFile() παράγει ένα φάκελο με το όνομα u1.predictions ο οποίος μέσα του περιλαμβάνει τα αποτελέσματα σε δύο αρχεία το part-00000 και το part-00001.

Εικόνα 2



Από κάτω φαίνονται μαζί τα αποτελέσματα από τα predictions files που δημιουργήθηκαν από το παράλληλο πρόγραμμα σε Java (αριστερά) στο Spark, το σειριακό πρόγραμμα σε Java (κέντρο) και από το σειριακό πρόγραμμα σε C++ (δεξιά) έτσι ώστε να επαληθευτούν τα αποτελέσματα. Παρατηρούμε ότι τα αποτελέσματα και στις 5 στήλες (customer id, movie id, rating, predicted rating, abs(rating - predicted rating)) είναι ίδια. Καθώς επίσης και ο συνολικός αριθμός των προβλέψεων όπως επίσης και ο μέσος όρος της απόλυτης διαφοράς του rating - predicted rating είναι ακριβώς ίδια.

Εικόνα 3



Επιπλέον ο λόγος που στο predictions file του παράλληλου πρόγραμματος μερικά στοιχεία της πέμπτης στήλης δεν είναι στοιχισμένα σωστά όπως τα υπόλοιπα της ίδιας στήλης, αποδίδεται στο γεγονός ότι υπάρχει μεγαλύτερη δεκαδική ακρίβεια στα νούμερα, ωστόσο τα αποτελέσματα που φαίνονται και στα 3 αρχεία είναι ίδια.

6. ΑΠΟΤΕΛΕΣΜΑΤΑ - ΑΞΙΟΛΟΓΗΣΗ

6.1 Μετρικές αξιολόγησης και συγκριτική αξιολόγηση

Ως μέτρο αξιολόγησης για την απόδοση των προγραμμάτων θεωρείται ο χρόνος και τα αποτελέσματα για τα οποία γίνεται αναφορά στην επόμενη παράγραφο. Ο συνολικός χρόνος εκτέλεσης για το σειριακό πρόγραμμα σε Java με το μικρό dataset του MovieLens των εκατό χιλιάδων βαθμολογιών ήταν 6 seconds. Αξιοσημείωτο είναι ότι ο χρόνος για την κατασκευή του αντικειμένου εγγινε, για το φόρτωμα του training file και το γράψιμο των αποτελεσμάτων στο testing file είναι ίσος με το μηδέν. ΄Ολος ο χρόνος εκτέλεσης του σειριακού προγράμματος εξαρτάται από τον υπολογισμό των features. Ενώ στο παράλληλο πρόγραμμα ο χρόνος εκτέλεσης για την κατασκευή του αντίστοιχου engine διαρκεί 5 seconds, για να διαβαστεί το training data η διάρκεια είναι 3 seconds, ο υπολογισμός των features 18 seconds και τελικά το γράψιμο των αποτελεσμάτων 1 second. Ουσιαστικά όλες οι μέθοδοι του σειριακού προγράμματος εκτελούνται πιο γρήγορα και με μεγάλη διαφορά. ΄Ενας λόγος στον οποίο μπορεί να αποδοθούν τα προηγούμενα αποτελέσματα είναι στο ότι το κόστος επικοινωνίας μεταξύ του driver και των executors είναι πολύ μεγάλο και ξεπερνάει κατά πολύ την χρονική πολυπλοκότητα των λειτουργιών και πράξεων που εκτελούνται στο σειριακό πρόγραμμα. ΄Ενας άλλος λόγος είναι το μέγεθος του dataset το οποίο είναι μικρό, γεγονός το οποίο δεν βοηθά στο να παραατηρηθεί κάποια βελτιστοποιήση από τη μεριά του παράλληλου προγράμματος. Είναι πιθανό ότι αν το πρόγραμμα ήταν να εκτελεστεί χρησιμοποιώντας το μεγαλύτερο dataset που υποστηρίζει το MovieLens για παράδειγμα αυτό των 20 εκατομμυρίων βαθμολογιών το οποίο είναι διακόσες φορές μεγαλύτερο από αυτό που χρησιμοποείται στη παρούσα εργασία και με την εκτέλεση του σε κατάλληλο περιβάλλον δηλαδή σε κάποιο computer cluster με υποδομή ευρείας κλίμακας τότε ίσως εκεί πιθανόν να παρατηρούνταν καλύτεροι χρόνοι στο παράλληλο πρόγραμμα γιατί τότε θα υπήρχαν περισσότεροι executors για να διαμοιραστούν τα tasks και επιπλέον το dataset θα ήταν πλέον αρκετά μεγάλο και πλέον ο φόρτος εργασίας θα αυξανόταν δραματικά για το ακολουθιακό πρόγραμμα, πράγμα που σημαίνει ότι ο χρόνος εκτέλεσης του θα αυξανόταν αρκετά. ΄Εγινε μεγάλος πειραματισμός με επιτυχία, μεγαλύτερου μεγέθους datasets όπως του ενός εκατομμυρίου βαθμολογιών, των τριών εκατομμυρίων βαθμολογιών και 5 εκατομμυριών βαθμολογιών αλλά και απόπειρα για εκτέλεση με μεγαλύτερα dataset χωρίς όμως επιτυχία για αυτά που ήταν πάνω από το όριο των 5 εκατομμυρίων, τον λόγο τον αναφέρω στην ενότητα 6.2 . Από τη συγκέντρωση των παραπάνω αποτελεσμάτων παρατηρήθηκαν τα εξής, πράγματι καθώς αυξάνεται το μέγεθος του dataset ο συνολικός χρόνος εκτέλεσης του παράλληλου προγράμματος είναι μικρότερος από ότι ο συνολικός χρόνος εκτέλεσης για το σειριακό πρόγραμμα σε Java, αυτό μπορεί να φανεί από τους χρόνους εκτέλεσης με dataset του ενός εκατομμυρίου και των πέντε εκατομμυρίων. Παρότι σε αυτά τα παραδείγματα σε όλες τις μεθόδους εκτός της CalcFeatures παρατηρείται ότι οι χρόνοι εκτέλεσης στο σειριακό πρόγραμμα είναι πιο γρήγοροι λόγω της μέγαλης διαφοράς στον χρόνο εκτέλεσης της CalcFeatures τελικά ο συνολικός χρόνος εκτέλεσης είναι πιο γρήγορος στο παράλληλο πρόγραμμα. Ωστόσο παρατήρησα ότι με dataset των 3 εκατομμυρίων που ανάμεσα στο 1 και 5 το σειριακό πρόγραμμα εκτελείται πιο γρήγορα, από αυτό συμπαιρένω ότι το πρόγραμμα δεν κάνει scale up ομαλά. ΄Επίσης ήθελα να τονίσω ότι η ανάπτυξη στο Apache Spark αποτέλεσε μεγάλη δυσκολία καταρχήν λόγω της άγνωστης πτυχής του για εμένα αλλά και εξαιτίας τη μη ύπαρξης εργαλείου debugger το οποίο περιόρισε σημαντικά τον τρόπο λειτουργίας μου αλλά και τον εξαναγκασμό μου να δημιουργήσω δικά μου βοηθητικά μηνύματα συνεχώς έτσι ώστε βηματικά να γίνεται η αποσφαλμάτωση του προγράμματος.

6.2 Αποτελέσματα

ΑΝΑΛΥΤΙΚΗ ΚΑΤΑΓΡΑΦΗ ΧΡΟΝΩΝ	ΣΕΙΡΙΑΚΟ ΠΡΟΓΡΑΜΜΑ C++	ΣΕΙΡΙΑΚΟ ΠΡΟΓΡΑΜΜΑ JAVA	ΠΑΡΑΛΛΗΛΟ ΠΡΟΓΡΑΜΜΑ JAVA, SPARK
Movie Lens Dataset 100K			
ENGINE CONSTRUCTION	0 ms	0 ms	5 136 ms
LOAD HISTORY	1 083 ms	0 ms	3 021 ms
CALCULATION FEATURE	41 246 ms	6 018 ms	18 281 ms
PROCESSING TEST	3 069 ms	0 ms	1 018 ms
TOTAL	**45 398 ms**	**6 018 ms**	**27 456 ms**
Movie Lens Dataset 1M			
ENGINE CONSTRUCTION	0 ms	0 ms	4 006 ms
LOAD HISTORY	2 033 ms	3 087 ms	8 012 ms
CALCULATION FEATURE	468 103 ms	170 324 ms	146 043 ms
PROCESSING TEST	3 008 ms	1 073 ms	7 105 ms
TOTAL	**473 144 ms**	**174 484 ms**	**165 166 ms**
Movie Lens Dataset 3M			
ENGINE CONSTRUCTION	0 ms	0 ms	8 007 ms
LOAD HISTORY	10 051 ms	7 083 ms	25 087 ms
CALCULATION FEATURE	1 297 037 ms	355 141 ms	374 032 ms
PROCESSING TEST	138 021 ms	3 067 ms	22 062 ms
TOTAL	**1 445 109 ms**	**365 291 ms**	**429 012 ms**
Movie Lens Dataset 5M			
ENGINE CONSTRUCTION	1 004 ms	0 ms	6 032 ms
LOAD HISTORY	15 043 ms	35 057 ms	51 071 ms
CALCULATION FEATURE	2 128 068 ms	1 173 314 ms	645 056 ms
PROCESSING TEST	12 057 ms	15 137 ms	48 053 ms
TOTAL	**2 156 172 ms**	**1 223 508 ms**	**750 212 ms**

΄Οπως μπορούμε να διαπιστώσουμε και από τις οθόνες εφαρμογής που προηγήθηκαν τα αποτελέσματα που παράχθηκαν στα αρχεία predictions files και από τα 2 προγράμματα σειριακό και παράλληλο σε Java είναι ακριβώς ίδια με το αρχείο με τα αποτελέσματα που έχουν παραχθεί από το πρόγραμμα σε C++. Τα αποτελέσματα γράφτηκαν στο αρχείο στην ίδια μορφή, σε 5 στήλες δηλαδή χωρισμένες με ένα tab ανάμεσά τους. Επίσης ο αριθμός των εξαγόμενων αποτελεσμάτων στα αρχεία είναι ο ίδιος αλλά και ο δείκτης μέσου όρου της απόλυτης τιμής της διαφοράς μεταξύ βαθμολογίας και εκτιμώμενης βαθμολογίας είναι ακριβώς ίδιος. Τα προηγούμενα είχαν τεθεί ως απαιτούμενα για την λειτουργικότητα των προγραμμάτων καθώς χωρίς σωστά αποτελέσματα όλες οι συγκρίσεις χρόνων δεν θα ήταν αξιόπιστες για να αποδωθεί προσοχή σε αυτές.

Στον παραπάνω πίνακα φαίνεται συγκεγκεντρωμένο το σύνολο των αποτελεσμάτων με διαφορετικού μεγέθους dataset.

Τα αποτελέσματα από πάνω παράχθηκαν μετά από διαδοχικές και επαναληπτικές εκτελέσεις κάθε προγράμματος έτσι ώστε να εξαχθεί ο μέσος όρος των αποτελεσμάτων όσο πιο σωστά γίνεται.

Το σειριακό πρόγραμμα σε Java και το παράλληλο πρόγραμμα σε Java στο Spark εκτελέστηκαν με τα ίδια resources στον προσωπικό μου υπολογιστή, όπως έχω προαναφέρει. Ωστόσο η εκτέλεση του προγράμματος σε C++ όπως επίσης προανέφερα έγινε σε υπολογιστή από τον χώρο εργασίας μου που έχει εγκατεστημένο Windows 7 λειτουργικό σύστημα και υποστηρίζει το Visual Studio 2013. Τα αναλυτικά χαρακτηριστικά των υπολογιστών έχουν προαναφερθεί στην ενότητα 2.3 .

Δοκιμάζοντας σε cluster mode να τρέξω το παράλληλο πρόγραμμα με dataset πάνω από 5 εκατομμύρια διαπίστωσα ότι δεν μπορεί να εκτελεστεί λόγω java.lang.OutOfMemoryError:Java heap space, δηλαδή δεν φτάνει η μνήμη στον driver για να το εκτελέσει. Ο driver έχει αρκετή memory συνολικά 24 GB RAM αλλά
μοιράζονται 4 και 4 στα δύο virtual machines. ΄Εγινε προσπάθεια για να γίνει η καλύτερη εφικτή οικονομία στα transformations, επίσης στη διαδικασία ανάπτυξης το ίδιο το Spark με ανάγκαζε να παραμείνω κάτω από τα 100 MB και με επίπονη προσπάθεια κατάφερα να περάσει οριακά. ΄Εγινε προσπάθεια και με τον Kryo Seriallizer αλλά δεν πέτυχε, στην επόμενη ενότητα έχω συμπεριλάβει μεταξύ άλλων και την προσπάθεια αυτή.

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

7.1 Σύνοψη της προτεινόμενης προσέγγισης με καινοτόμα στοιχεία και θετικά αποτελέσματα

Τα θετικά αποτελέσματα εν κατακλείδι που προέκυψαν από την παρούσα πτυχιακή εργασία είναι πολλαπλά. Αρχικά έγινε επιτυχημένα η παραμετροποιήση του σειριακού αλγορίθμου στην ίδια γλώσσα προγραμματισμού που είχε υλοποιηθεί αρχικά (C++) έτσι όπως δίνεται από τον ιστότοπο http://www.timelydeve opment.com/demos/NetflixPrize.aspx, (ο οποίος μάλιστα βραβεύτηκε ως ο 3ος καλύτερος στο σχετικό διαγωνισμό του Netflix). Η παραμετροποιήση έγινε ώστε ο νέος αλγόριθμος να δέχεται ως είσοδο τα dataset στη μορφή του MovieLens. 'Επειτα έγινε μετατροπή του τελευταίου προαναφερμένου προγράμματος σε γλώσσα προγραμματισμού Java επίσης σε σειριακή μορφή. Ο χρόνος εκτέλεσης του ακολουθιακού προγράμματος σε Java με το μικρό dataset των 100.000 βαθμολογιών ήταν αρκετά μικρός 6 second. Πράγμα το οποίο δεν ήταν πολύ ενθαρυντικό για τον χρόνο που θα πετύχει το παράλληλο πρόγραμμα. Καθώς εάν παρατηρηθεί βελτιστοποίηση στο συνολικό χρόνο ή σε κάποια μέθοδο ξεχωριστά αυτό θα συμβεί μόνο στα μεγαλύτερα dataset του MovieLens αυτά των 10.000.000 βαθμολογιών και 20.000.000 βαθμολογιών. Προχωρόντας, έγινε εφικτή η παραλληλοποίηση του αλγορίθμου (με εξαίρεση τη μέθοδο CalcFeatures() της οποίας η εκτέλεση όπως εξηγώ αναλυτικά στην αμέσως επόμενη ενότητα δεν μπορεί να παραλληλοποιηθεί σωστά στο Apache Spark) στο βαθμό βέβαια που επέτρεπε η ίδια η φύση του αλγορίθμου αλλά και οι δομές δεδομένων για παράλληλη επεξεργασία που υποστηρίζει το Apache Spark, πράγμα το οποίο είναι θετικό από μόνο του. Ο συνολικός χρόνος εκτέλεσης κυμαινόταν μεταξύ 25 και 28 seconds χρησιμοποιώντας το μικρό dataset. Για τα 3 προηγούμενα προγράμματα τα αποτελέσματα που εξάγονταν ήταν σε παρόμοια μορφή στα αντίστοιχα predictions files και όπως αποδείχτηκε είναι όλα ίδια. Επίσης ο συνολικός αριθμός αποτελεσμάτων είναι ο ίδιος αλλά και ο δείκτης μέσου όρου της απόλυτης τιμής της διαφοράς μεταξύ του rating και predict rating είναι και αυτός ίδιος. Τα προηγούμενα είχαν τεθεί σαν στόχοι μερικά για απαιτούμενη λειτουργικότητα και άλλα για να ερευνηθούν.

7.2 Δυσκολίες που παρουσιάστηκαν

Οι περιορισμοί αυτοί δημιούργησαν δυσκολίες στην παραλληλοποίηση ορισμένου τμήματος του αλγο ρίθμου. Χαρακτηριστικό παράδειγμα δυσκολίας ήταν ότι δεν υπήρχε παραλληλοποιήσιμη R/W δομή που να υποστηρίζει δισδιάστατους πίνακες. Αυτό οφείλεται στην immutable(αμετάβλητη) φύση των RDD που είναι και η βασική παραλληλοποιήσιμη δομή στο Spark. Ωστόσο έγιναν απόπειρες υλοποίησης της παραλληλοποίησης με δικής μου έμπνευσης κλάσεις. Ως λύση στο προηγούμενο πρόβλημα, έγινε προσπάθεια χρήσης του πακέτου amplab:spark-indexedrdd:0.3 (IndexedRDD) που υπόσχεται τα ακόλουθα: 'IndexedRDD is an updatable key-value store for Spark. It enables efficient keyed lookups, updates, deletions and joins for key-value pair RDDs¨. Δυστυχώς είναι συμβατό μόνο με την πολύ παλαιά έκδοση, 1.5.0 του Spark και χρησιμοποιεί ancillary features εκείνης της έκδοσης τα οποία δεν υποστηρίζονται πλέον από το 1.6.0 και 1.6.1. Θεωρούνται απαρχαιωμένα.

Επίσης ερευνήθηκε η βιβλιοθήκη MLib (Machine Learning Library) η οποία πέραν των έτοιμων συναρτήσεων για Dimensionality Reduction όπως την SVD, παρέχει 4 εκδοχές distributed matrix με σημαντικούς περιορισμούς η κάθε μια («We assume that the number of columns is not huge for a RowMatrix», «A CoordinateMatrix should be used only when both dimensions of the matrix are huge and the matrix is very sparse», κτλ) και κοινό χαρακτηριστικό τους ότι δεν επιτρέπουν την ενημέρωση κάποιου κελιού, όπως απαιτείται από την CalcFeatures().

Επιπρόσθετα, ερευνήθηκε η βιβλιοθήκη DataFrames and SQL, που επίσης έχουν πρόβλημα στην ενημέρωση των δεδομένων εξαιτίας του ότι βασίζονται πάνω στα immutable RDDs.

Αυτά σε συνδυασμό με τους περιορισμούς παραλληλοποίησης του αλγορίθμου για την μέθοδο CalcFeatures() αποτέλεσε εμπόδιο για την πλήρη παραλληλοποίηση του προγράμματος. Πιο αναλυτικά η μέθοδος CalcFeatures() δεν μπόρεσε να παραλληλοποιηθεί για τους εξής λόγους: η κυρίως εργασία του αλγορίθμου γίνεται με τη μέθοδο CalcFeatures() που σκοπό έχει τον υπολογισμό των τιμών στους 2 πίνακες CustomerFeatures και MovieFeatures. Οι τιμές στα κελιά των πινάκων αυτών ενημερώνονται κατά κύματα (epochs) μέχρι να επιτευχθεί κάποιο αποδεκτό συνολικό σφάλμα. Η εργασία αυτή αποτελείται από 3 ένθετους βρόχους, με κατ' ελάχιστο εκτελέσεις των εντολών του εσωτερικού βρόχου να είναι $10^5 * 120 * 64$.

Οι εντολές αυτές είναι κυρίως οι 4 βασικές πράξεις αριθμητικής και ανάγνωση και εγγραφή σε 2 δισδιάστατους και έναν μονοδιάστατο πίνακα.

Παρόλα αυτά έγινε και μια απόπειρα για παραλληλοποίηση της CalcFeatures η οποία απέτυχε για τους εξής λόγους που παρατήρησα και σημείωσα κατά τη διαδικασία.

1. Θεωρεί ως βασικές εισόδους πέραν κάποιων σταθερών, τον πίνακα με όλα τα rating (custId, movieId, rating, με αρχικοποιημένη την cache) και τις 2 κατάλληλα αρχικοποιημένες μήτρες customer και

movie feature.

2. Ως βασικές εξόδους έχει τις μήτρες customer και movie feature.

3. Με την ολοκλήρωση υπολογισμού κάθε feature (εξωτερικός βρόχος) γίνεται ενημέρωση των cache που αφορά το κάθε rating, έτσι ώστε να χρησιμοποιηθεί στους υπολογισμούς των επόμενων feature. ΄Αρα ο εξωτερικός βρόχος δεν επιτρέπει να υπολογισθούν τα feature ανεξάρτητα το ένα από το άλλο και άρα δεν μπορεί να παραλληλοποιηθεί.

4. Οι επαναλήψεις του ενδιάμεσου βρόχου που αφορά τα epochs, δεν είναι deterministic (τετελεσμένες) για να μπορεί να παραλληλοποιηθεί με RDD. Αυτό συμβαίνει γιατί μέρος της συνθήκης τερματισμού του βρόγχου, επηρεάζεται από τους υπολογισμούς μέσα στο βρόγχο (rmse).

5. Ο μόνος βρόγχος ως πιθανά επιδεχόμενος παραλληλοποίησης είναι ο εσωτερικός βρόγχος για όλα τα rating ενός epoch. Το πλήθος των επαναλήψεων είναι μεγάλο, άρα υπάρχει ευκαιρία για κέρδος λόγω παράλληλης εκτέλεσης. Επίσης η κάθε επανάληψη χρησιμοποιεί πολλά σταθερά δεδομένα, όπως το feature id, το custId, movieId, rating, cache, επίσης μπορεί να διαβάσει τις τιμές cust_feature[f][custId] και movie_feature[f][movieId] από κάποιο RDD. Τα αναγκαία δεδομένα των cust_feature και movie_feature είναι μόνο οι συγκεκριμένες γραμμές με index το feature id. Οι υπολογισμοί του βρόγχου αυτού άρα, θα μπορούσαν να γίνουν μέσω ενός transformation πάνω σε ένα RDD το οποίο θα περιείχε τα rating record (custId, movieId, rating, cache), 2 RDD βασισμένα στις γραμμές cust_feature[f][] και movie_feature[f][] τα οποία θα δημιουργούνται πριν το transformation.

6. ΄Ενα ζήτημα που χρειάζεται απάντηση είναι το πώς θα μπορέσει το transformation που θα κάνει τους υπολογισμούς να έχει πρόσβαση στην τιμή του feature id. Αυτό δεν είναι δυνατό με lambda expression ή anonymous class γιατί απαιτεί τέτοιες τιμές να βρίσκονται σε μεταβλητές τύπου final ή effectively final. Λύση δόθηκε με τη δημιουργία Inner Class (non-static nested class) που κάνει implement το κατάλληλο interface (call() function) με data member το feature id, δημιουργία αντικειμένου στην αρχή της CalcFeatures() και ενημέρωση του data member μέσω κλήσης του setter σε κάθε αλλαγή του feature id στον εξωτερικό βρόγχο. ΄Ετσι η συνάρτηση call() που παρέχει την υλοποίηση του interface, έχει σε κάθε κλήση τη σωστή τιμή του feature id.

7. Άλλο ζήτημα που χρειάστηκε απάντηση είναι ότι η parallelize μέθοδος του javaSparkContext απαιτεί μονοδιάστατο πίνακα (vector) από αντικείμενα προκειμένου να δημιουργήσει RDD. ΄Αρα οι μήτρες

cust_feature[][] και movie_feature[][] θα έπρεπε να αναπαρασταθούν ως RDD<MatrixRow>, όπου το MatrixRow θα είναι μια custom class που θα αναπαριστά μια γραμμή από αριθμούς ως List<float>.

8. Λόγω του μεγάλου μεγέθους των ανταλλασσόμενων δεδομένων μεταξύ του Driver και των Executors, χρειάστηκε να γίνει και χρήση του εναλλακτικού Kryo Serializer ο οποίος έχει καλύτερες επιδόσεις από τον default serializer του Spark. Γι' αυτό το σκοπό υλοποιήθηκε custom Kryo Registrator όπου δηλώθηκαν όλες οι κλάσσεις που επιθυμούμε να γίνονται serialize μέσω Kryo. Τελικά όμως στο τελικό παράλληλο πρόγραμμα αυτό που τρέχει αλλά εκτελεί σειριακά την CalcFeatures, δεν χρησιμοποιήθηκε καθώς είχε συνεχώς προβλήματα με serialization.

9. Ένα ζήτημα ακόμη ήταν το πώς θα επιτύχει το parallel transformation τα ίδια output με αυτά του σειριακού βρόγχου. Το sq αρχικοποιείται πριν το βρόγχο και μέσα στο βρόγχο αυξάνεται διαδοχικά. Αυτό επιτυγχάνεται δηλώνοντας το sq ως Accumulator, τον οποίο ο Driver αρχικοποιεί πριν την κλήση του transformation , ενώ οι Executors τον αυξάνουν μέσω του κώδικα της transformation function.

10. Το δυσκολότερο ζήτημα που έπρεπε να λυθεί ήταν αυτό της ενημέρωσης των τιμών στα customer και movie feature RDD<MatrixRow>, στις αντίστοιχες θέσεις που θα γίνονταν στις μήτρες cust_feature[f][custId] και movie_feature[f][movieId]. Η μόνη λύση που επέτρεπε την ενημέρωση αυτή ήταν η δημιουργία custom Accumulator για το MatrixRow. ημιουργήθηκε η class MatrixRowAccumulatorParam που κάνει implement το AccumulatorParam<MatrixRow> interface και η οποία επιτρέπει να προστεθούν 2 MatrixRow. Έτσι πριν το transformation αρχικοποιούνται 2 Accumulator<MatrixRow> ένας για κάθε γραμμή cust_feature[f][] και movie_feature[f][] με τις τιμές των μητρών και μέσα στην transformation function δημιουργούνταν ένα νέο MatrixRow με μηδενικά όλα τα στοιχεία εκτός από τη στήλη custId ή movieID (ανάλογα) και κλήση της Accumulator.add() για να κάνει την ενημέρωση. Μετά το transformation γίνεται ενημέρωση των μητρών από τους Accumulator<MatrixRow>.

Δυστυχώς οι υπολογισμοί με αυτήν την απόπειρα παραλληλοποίησης ενός κατά τα άλλα σειριακού αλγορίθμου δεν έδωσε σωστά αποτελέσματα και δεν είχε ικανοποιητικές επιδόσεις. Έτσι παραλληλοποιήθηκαν τελικά όλες οι άλλες function εκτός βέβαια από την μέθοδο CaclFeatures() στην οποία ο υπολογισμός των Features γίνεται ακολουθιακά στο πρόγραμμα.

Από κάτω ακολουθούν τμήματα κώδικα με τις προσπάθειες που έγιναν.

Ακολουθεί δοκιμαστική υλοποίηση του **KryoRegistrator**.

Ο Kryo είναι ένα γρήγορο serialization framework για την Java. Στόχος του είναι η επίτευξη ταχύτητας και αποτελεσματικότητας και είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου τα αντικείμενα πρέπει να είναι persistent. Persistent αντικείμενα είναι αυτά που δίνουν μια ένδειξη ότι η κατάσταση ενός αντικειμένου θα αποθηκευτεί μόνιμα, ακόμη και μετά την εκτέλεση του προγράμματος.

Ουσιαστικά αυτό που έγινε ήταν η δημιουργία μιας Kryo registrator δικιάς μου κλάσης και επίσης έγινε ρύθμιση του serializable τύπου και της Kryo registrator κλάσης στο spark conf.

MyClassRegistrator

```
// custom registrator for Kryo Serializer - Static Nested class
public static class MyClassRegistrator implements KryoRegistrator {
@Override
public void registerClasses(Kryo kryo)
kryo.register(Customer.class, new FieldSerializer(kryo, Customer.class));
kryo.register(Movie.class, new FieldSerializer(kryo, Movie.class));
kryo.register(SparkData.class, new FieldSerializer(kryo, SparkData.class));
kryo.register(TrainingData.class, new FieldSerializer(kryo, TrainingData.class));
kryo.register(TestingData.class, new FieldSerializer(kryo, TestingData.class));
kryo.register(MatrixRow.class,new FieldSerializer(kryo, MatrixRow.class));
kryo.register(MatrixRowAccumulatorParam.class,new FieldSerializer(kryo,MatrixRowAccumulatorParam.
class));
kryo.register(SVDMovieLensSparkJava.class, new FieldSerializer(kryo, SVDMovieLensSparkJava.class));
}
}
}
```

Ακολουθεί η κλάση **MatrixRow**

Στην κλάση MatrixRow γίνεται δήλωση του πεδίου row το οποίο αποτελεί μια λίστα από Double αντικείμενα, σειρά έχει ο constructor και ο getter για το πεδίο. Στην συνέχεια ακολουθεί η μέθοδος size που επιστρέφει το μέγεθος της λίστας. Τελικά με την toString γίνεται εκτύπωση της λίστας.

MatrixRow

```
package edu.berkeley.svdmovielens;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MatrixRow implements Serializable    {

    // Data Members
    List<Double> row;


    // Constructor
    public MatrixRow(List<Double> row) {
        this.row = row;
    }

    // Data Accessors
    public List<Double> getRow() {
        return  row;
    }


    // Methods
    public int size()  {
        return  this.row.size();
    }
```

```
    @Override
    public String toString() { String
        s = "|-<";
        s = s + String.valueOf(this.row.size()) + ">";
        Double F;
        Iterator<Double> it = this.row.iterator(); while
        (it.hasNext()) {
            F = it.next();
            s += (F.toString() + "\t");
        }
        return s;
    }
}
```

Ακολουθεί η κλάση **MatrixRowAccumulatorParam**.

Η κλάση MatrixRowAccumulatorParam κληρονομεί τις ιδιότητες της κλάσης AccumulatorParam, η οποία είναι ένα helper αντικείμενο για τον καθορισμό του τρόπου που θα γίνονται accumulate οι τιμές ενός συγκεκριμένου τύπου, στην προκειμένη περίπτωση MatrixRow αντικειμένου. Υλοποιεί τρεις μεθόδους η κλάση. Πρώτη μέθοδος είναι η zero η οποία επιστρέφει την μηδενική τιμή ¨ταυτότητα' για έναν accumulator τύπο, δεδομένου της αρχικής τιμής που περνιέται ως παράμετρος στη μέθοδο. εύτερη μέθοδος είναι η addInPlace η οποία παίρνει ως παραμέτρους δύο σύνολα t και t1 από accumulated δεδομένα. Ουσιαστικά η μέθοδος αυτή επιστρέφει τη συγχώνευση των δύο συσσωρευμένων αξιών και έχει τη δυνατότητα να τροποποιήσει και να επιστρέψει την πρώτη τιμή για επίτευξη απόδοσης. Τρίτη μέθοδος είναι η addAccumulator η οποία παίρνει ως παραμέτρους την τρέχουσα τιμή του accumulator και τα δεδομένα που θα προστεθούν στον accumulator. Αυτό που επιστρέφεται είναι η νέα τιμή του accumulator μετά την πρόσθεση.

MatrixRowAccumulatorParam

```
package edu.berkeley.svdmovielens;
import java.util.ArrayList;

import  java.util.List;

import   org.apache.spark.AccumulatorParam;


class MatrixRowAccumulatorParam implements AccumulatorParam<MatrixRow> {


    @Override
    public MatrixRow zero(MatrixRow initialValue) { List<Double>  lst  =
        new ArrayList<>();
        for (int i = 0; i < initialValue.size(); i++) { lst.add(0.0);
        }
        return  new MatrixRow(lst);
    }




    @Override
    public MatrixRow addInPlace(MatrixRow t, MatrixRow t1) { return
        addAccumulator(t, t1);
    }






    @Override
    public MatrixRow addAccumulator(MatrixRow t,  MatrixRow t1) {
        // need to create a brand new List<> and MatrixRow in order to work!!! List<Double> newLst =
        new  ArrayList<>(),
                    tlst = t.getRow(), t1lst =
                    t1.getRow();
        for (int i=0; i<t.size(); i++) { newLst.add(tlst.get(i)  +
```

```
            t1lst.get(i));
        }
        return new MatrixRow(newLst);
    }

}
```

Ακολουθεί η **inner** κλάση **InnerFunctionClass3**.

Η κλάση κάνει implement ένα PairFuncion για την κατασκευή ενός PairRDD. Το PairFuncion από τον ορισμό του παίρνει τρεις παραμέτρους, πρώτη παραμετρος είναι ένα Tuple2 από Integer στη θέση του key και στη θέση του value ένα Tuple3 από Data, MatrixRow και MatrixRow, δεύτερη παράμετρος είναι ένας Integer και τρίτη παράμετρος ένα Tuple3 από Data, MatrixRow και MatrixRow, έτσι ώστε τελικά να επιστραφεί ένα καινούριο key-value pair Tuple2 με την δεύτερη και τρίτη παράμετρο που δόθηκαν αρχικά. Ως πεδίο δηλώνεται ο ακέραιος feature, έπειτα ακολουθεί ο constructor της InnerFunctionClass3 και ο setter του feature. Στην συνέχεια υλοποιείται η μόνη μέθοδος που υποστηρίζει η PairFunction που είναι η call η οποία είναι αυτή που θα επιστρέψει το Tuple2 από Integer και Tuple3<Data, MatrixRow, MatrixRow>. Μέσα στη μέθοδο δηλώνονται οι μεταβλητές index στην οποία ανατίθεται η τιμή του key από το Tuple2 που παιρνιέται ως παράμετρος στη μέθοδο, έπειτα έχουμε την δήλωση των κωδικών ταινίας, πελάτη, βαθμολογίας και cache οι οποίες παίρνουν τις τιμές τους από τα πεδία του data αντικειμένου, επιπλέον δηλώνονται το movieRow και customerRow τα οποία παίρνουν τιμές κάνοντας πρόσβαση στο value του Tuple2 που παιρνιέται ως παράμετρος και συγκεκριμένα στη δεύτερη και τρίτη θέση του Tuple3, δηλαδή τα δύο MatrixRow που είναι οι λίστες με τα features για customer και movie για το συγκεκριμένο feature οι οποίες έπειτα ανατίθενται στο cf και mf αντίστοιχα. Μετέπειτα δηλώνεται και αρχικοποιείται με τιμή ένα, ο αθροιστής sum και ξεκινάει ένας επαναληπτικός βρόγχος που διατρέχει όλα τα feature κατά την διάρκεια του οποίου έγινε προσπάθεια να υπολογιστεί το sum για κάθε feature. Μετά τον επαναληπτικό βρόγχο δημιουργείται ένα Tuple3 το t3 με data, customerRow και movieRow το οποίο θα αποτελέσει το value πεδίο για το Tuple2 που θα επιστραφεί από τη μέθοδο με index που θα επιστραφεί .

InnerFunctionClass3

```
class InnerFunctionClass3 implements PairFunction<Tuple2<Integer,Tuple3<Data,MatrixRow,MatrixRow>>,
Integer,Tuple3<Data,MatrixRow,MatrixRow>>{

        // data members
        int feature;
```

```
        public  InnerFunctionClass3()  {
        }

        // data accessors
        public void setFeature(int feature) {
            this.feature   =   feature;
      }

    @Override
public Tuple2<Integer,Tuple3<Data,MatrixRow,MatrixRow>> call(Tuple2<Integer,
Tuple3<Data,MatrixRow,MatrixRow>> tuple) throws Exception{
    Integer index = tuple._1;
    Data data = tuple._2._1();
    int movieId = data.MovieId;
    int  custId  =  data.CustId;
    int rating = data.Rating;
    float cache = data.Cache;
    MatrixRow movieRow = tuple._2._3();
    MatrixRow customerRow  =  tuple._2._2();
    float   mf   =  movieRow.getRow().get(movieId);
    float cf = customerRow.getRow().get(custId);

    double sum =  1;

    for (int f = 0; f < MAX_FEATURES; f++)   {

            sum += this.m_aMovieFeatures[f][movieId] * this.m_aCustFeatures[f][custId];
            if (sum  >  5) {
                 sum  = 5;
            }
            if (sum  <  1) {
```

```
                    sum  = 1;
                }
            }
            Tuple3<Data, MatrixRow, MatrixRow> t3 = new Tuple3<>(data, customerRow, movieRow);
            return  new  Tuple2<>(index,   t3);
        }
    }
```

Ακολουθεί η απόπειρα για παραλληλοποίηση της μεθόδου **CalcFeatures**().

Αρχικά γίνονται οι δηλώσεις των πεδίων που υπήρχαν στην αντίστοιχη υλοποίηση της σειριακής εκδοχής του CalcFeatures() και επιπλέον δηλώνονται οι μεταβλητές MIN_EPOCHS και MIN_IMPROVEMENT οι οποίες παίρνουν τις τιμές των αντίστοιχων broadcasted πεδίων. ΄Επειτα ακολουθεί η δήλωση των μεταβλητών της inner κλάσης που χρησιμοποιούνται στα transformations. ΄Επειτα ακολούθησε η δημιουργία ενός RDD με Data και δύο MatrixRows για τους πίνακες CustomerFeature[f][] και MovieFeature[f][]. Στην συνέχεια γίνεται η δημιουργία δύο broadcast μεταβλητών για τα MatrixRows του customer και movie έτσι ώστε να έχουν πρόσβαση στις παλιές τιμές (cache). ΄Υστερα ακολουθεί η δημιουργία δύο accumulators για τα MatrixRows του customer και movie. Γίνεται απόπειρα να υπολογιστεί το sq. Συλλέγονται οι βαθμολογίες με transformation, βρίσκεται το πλήθος τω βαθμολογιών, γίνεται απόπειρα να υπολογιστεί η ρίζα του μέσου τετραγωνικού σφάλματος. Τυπώνονται τα αποτελέσματα παρόλα αυτά η διαφορά του rmse_last και του rmse δεν έβγαζε σωστά αποτελέσμτα με τα αντίστοιχα στο σειριακό πρόγραμμα (χρήση debugger για το σειριακό) και εκτύπωση των τιμών στο παράλληλο πρόγραμμα για να συγκριθούν.

spark_CalcFeatures

```
 void spark_CalcFeatures() {
int  f,  e;
double /*err,*/ rmse_last = 0.0, rmse = 2.0;
int movieId;
double  cf, mf;
```

```
long tmp;
double MIN_EPOCHS = this.MIN_EPOCHS_BRDCST.getValue();
double MIN_IMPROVEMENT = this.MIN_IMPROVEMENT_BRDCST.getValue();

// declare inner class variables to use in tranformations
CalculateSQ_InnerClass funcCalcSQ = new CalculateSQ_InnerClass();
for (f = 0; f < /*this.MAX_FEATURES*/ 1; f++) {
Message("Calculating feature " + f);
// Create an RDD with Data and the MatrixRows from CustomerFeature[f][] and MovieFeature[f][] matrices
funcCalcSQ.setFeature(f);
for (e = 0; /*(e < MIN_EPOCHS) || (rmse <= rmse_last - MIN_IMPROVEMENT)*/ e < 10; e++) {
// Create 2 Broadcast variables for the Customer
//and Movie MatrixRows to have access to previous values
customerFeatureMatrixRow_BRDCST = sc.broadcast(mapCustomerFeatures.get(f));
movieFeatureMatrixRow_BRDCST = sc.broadcast(mapMovieFeatures.get(f));
// Create the 2 Accumulators for the Customer and Movie MatrixRows
customerFeatureMatrixRow_ACC=sc.accumulator(mapCustomerFeatures.get(f),new MatrixRowAccumulatorParam())
movieFeatureMatrixRow_ACC = sc.accumulator(mapMovieFeatures.get(f), new MatrixRowAccumulatorParam());
this.sq_ACC = sc.accumulator(0.0);
rmse_last = rmse;
m_aTrainingRatings_PairRDD = m_aTrainingRatings_PairRDD.mapToPair(funcCalcSQ);
//m_aRatings.collect();
tmp = m_aTrainingRatings_PairRDD.count();
rmse = Math.sqrt(this.sq_ACC.value() / m_nRatingCount_BRDCST.getValue());
// Update the maps with the accumulated MatrixRows
mapCustomerFeatures.put(f, customerFeatureMatrixRow_ACC.value());
mapMovieFeatures.put(f, movieFeatureMatrixRow_ACC.value());
System.out.printf("|---------> Epoch = %d\tRmse_last-Rmse = %f\n", e, rmse_last - rmse);
}

Message("Customer ACCUMULATOR after transformation for Feature " +
```

```
+ f + ", Epoch " + e + "\n" +  customerFeatureMatrixRow_ACC.value().toString());
//System.out.printf("|--> custmatrowAccum row 0 is %s\n",
// customerFeatureMatrixRow_ACC.value().toString());

// Re-Calculate cache values
//spark_m_FeatureDataRows  =  spark_m_FeatureDataRows.mapToPair(funcCalcCaches);
      }
}
```

Ακολουθεί η **inner** κλάση **CalculateSQ_InnerClass**.

Η κλάση CalculateSQ_InnerClass αποτελεί μια άλλη απόπειρα να υλοποιηθεί η InnerFunctionClass3 μόνο που αυτή τη φορά η κλάση υλοποιεί PairFunction με πρώτη παράμετρο ένα Tuple2 με key Integer και value TrainingData , δεύτερη παράμετρο Integer και τρίτη παράμετρος TrainingData. Η TrainingData κλάση έχει δηλωμένα ως πεδία custId, MovieId, Rating και Cache. Παρόμοια με την προηγούμενη απόπειρα υλοποίησης της inner κλάσης, δηλώνεται το πεδίο feature και ο αντίστοιχος setter του, όπως επίσης ο default constructor της κλάσης. Αντίστοιχα υλοποιείται η μέθοδος call, γίνεται δήλωση των πεδίων και παίρνουν τις ίδιες τιμές με την προηγούμενη υλοποίηση της inner κλάσης με εξαίρεση την δήλωση των δύο MatrixRow μεταβλητών και τις τιμές οπυ ανατίθενται στο cf και mf. Στην συνέχεια υπολογίζεται το άθροισμα movie feature και customer feature και στην συνέχεια παιρνάει από τους ελέγχους έτσι ώστε να περιοριστεί η βαθμολογία μεταξύ των τιμών ένα έως πέντε. ΄Επειτα ακολουθεί η ίδια διαδικασία για το trailing και τελικά υπολογίζεται το err και προσθέτουμε το τετράγωνο του error στον accumulator του sq. Και αφού δεν γίνεται να ενημερωθεί μια συγκεκριμένη τιμή σε ένα MatrixRow accumulator δοκιμάζω να κατασκευάσω ένα μηδενικό MatrixRow, ουσιαστικά έγινε μια ανορθόδοξη προσπάθεια θα ενημερώνεται ο MatrixRow Accumulator ενημερώνοντας ένα μεμονωμένο στοιχείο και παιρνώντας αυτό σε ένα άδειο MatrixRow. Ουσιαστικά ακόμη και αν λειτουργούσε αυτή η μέθοδος θα έπρεπε για κάθε στοιχείο να γίνεται δημιουργία μια λίστας από double πράγμα το οποίο θα ήταν αναποτελεσματικό καθώς πλέον το κόστος της επικοινωνίας θα ήταν αρκετά μεγάλο. Το ίδιο πράγμα που δοκίμασα για τους customers δοκίμασα να κάνω και για τα movies και στην συνέχεια να επιστραφεί το tuple από τη μέθοδο. Το πρόβλημα στη περίπτωση αυτή πέρα από την πολύ κακή παραλληλοποίηση είναι ότι στη σειριακή έκδοση της CalcFeatures στο τέλος του πρώτου εξωτερικού επαναληπτικού βρόγχου ο οποίος διατρέχει τα features γίνονται cache off τα παλιά predictions για να γίνει αυτό χρειάζεται πρόσβαση στις προηγούμενες cache τιμές το οποίο δεν μπορεί να επιτευχθεί παράλληλα σε συνδυασμό με την πολυπλοκότητα και την αλληλοεξάρτηση των μεταβλητών με τους τρεις επαναληπτικούς βρόγχους και τις πράξεις και υπολογισμούς που υλοποιούνται μέσα σε αυτούς.

CalculateSQ_InnerClass

```
class CalculateSQ_InnerClass implements PairFunction<Tuple2<Integer, TrainingData>,
Integer, TrainingData> {

        // data members
        int feature;

        // constructor
        public  CalculateSQ_InnerClass() {
        }

        // data accessors
        public void setFeature(int feature) {
            this.feature   =   feature;
        }

        // implementation of interface obligations
        @Override
    public Tuple2<Integer, TrainingData> call(Tuple2<Integer, TrainingData> tuple) throws Exception{
            Integer  index  =   tuple._1;
            TrainingData data = tuple._2;
            int movieId = data.MovieId;
            int  custId  =  data.CustId;
            int rating = data.Rating;
            double cache =  data.Cache;
            double  cf  =  customerFeatureMatrixRow_BRDCST.getValue().row.get(custId  -  1);
```

```
double mf = movieFeatureMatrixRow_BRDCST.getValue().row.get(movieId - 1);
/* cache broadcast values in local variables to avoid calling them
   repeatedly and to make code easier to read */
double INIT = SVDMovieLensSparkJava.this.INIT_BRDCST.getValue();
double LRATE = SVDMovieLensSparkJava.this.LRATE_BRDCST.getValue();
double K = SVDMovieLensSparkJava.this.K_BRDCST.getValue();
double sum = (cache > 0) ? cache : 1;
// Add contribution of current feature
sum += mf * cf;
if (sum > 5) {
     sum = 5;
}
if (sum < 1) {
     sum = 1;
}
// Add up trailing defaults  values
sum += (MAX_FEATURES_BRDCST.getValue() - feature - 1) * (INIT * INIT);
if  (sum  >  5) {
     sum  = 5;
}
if (sum < 1) {
     sum = 1;
}
double err = (1.0 * rating - sum);
sq_ACC.add(err * err);
/* I cannot update a single value in a MatrixRow Accumulator, so I
   have to construct an "zero" MatrixRow, update a single item and
   add this MatrixRow to the MatrixRow Accumulator!!
 */
List<Double> custlst = new ArrayList<>();
for (int i = 0; i < customerFeatureMatrixRow_BRDCST.getValue().row.size(); i++) {
```

```
                custlst.add(0.0);
            }
            custlst.set(custId - 1, LRATE * (err * mf - K * cf));
            if (feature == 1) {
Message("CalculateSQ_InnerClass.call()-custFeatureRow("+feature+","+custId+")="+custlst.get(custId-1));
            }
            customerFeatureMatrixRow_ACC.add(new MatrixRow(custlst));
            List<Double> movielst = new ArrayList<>();
            for (int i = 0; i < movieFeatureMatrixRow_BRDCST.getValue().row.size(); i++) {
                movielst.add(0.0);
            }
            movielst.set(movieId - 1, LRATE * (err * cf - K * mf));
            movieFeatureMatrixRow_ACC.add(new MatrixRow(movielst));
            return tuple;
        }
    }
```

Ακολουθεί η παράλληλη εκδοχή της **PredictRating()**

Οι αλλαγές που έγιναν σε αυτή την προσέγγιση είναι στις δύο αθροίσεις με τη μεταβλητή sum. Αρχικά στην πρόσθεση της συμμετοχής του τωρινού feature, πλέον δεν παιρνιέται σαν παράμετρος το movie feature και customer feature αλλά οι κωδικοί των ταινιών και πελατών οι οποιοί στην συνέχεια χρησιμοποιούνται ως δείτκες στους πίνακες m_aMovieFeatures και m_aCustFeatures που έχουν τα feature ανά ταινία και πελάτη. Και στην πρόσθεση των default τιμών του trailing η μόνη διαφορά είναι ότι αντί να χρησιμοποιηθούν τα σειριακά data members χρησιμοποιούνται τα αντίστοιχα broadcast πεδία που δηλώνονται στο παράλληλο πρόγραμμα.

spark_PredictRating()

```
double spark_PredictRating(double movieFeature,double customerFeature,int feature,
double cache,boolean bTrailing){
// Get cached value for old features or default to an   average
```

```
double sum = (cache > 0) ? cache : 1;
// Add contribution of current feature
sum += movieFeature * customerFeature;
if (sum > 5) {
        sum = 5;
      }
        if (sum < 1) {
            sum = 1;
         }
         // Add up trailing defaults values
         if (bTrailing) {
sum+=(MAX_FEATURES_BRDCST.getValue()-(feature+1)-1)*(INIT_BRDCST.getValue()*INIT_BRDCST.getValue());
              if (sum > 5) {
                   sum = 5;
              }
              if (sum < 1) {
                   sum = 1;
              }
         }
         return sum;
 }
```

7.3 Πιθανός χώρος για βελτιώσεις

Πιθανή βελτίωση στα προγράμματα που υλοποιήθηκαν προηγουμένως και στο σειριακό και στο παράλληλο, είναι η μετατροπή όλων των τύπων δεδομένων που δηλώθηκαν ως double σε float. Υπάρχει η άποψη ό,τι οι τύποι δεδομένων double χρειάζονται περισσότερο χώρο στη μνήμη από ό,τι οι τύποι δεδομένων float. Υπάρχει βέβαια και η άποψη ότι η επιλογή μεταξύ float και double δεν είναι τόσο σημαντική όσο ο τρόπος που χρησιμοποιούνται τα δεδομένα. Εάν θέλουμε να κάνουμε μικρούς υπολογισμούς σε μεγάλο dataset ένας μικρός τύπος δεδομένων όπως ο float είναι προτιμότερος. Ενώ η υλοποιήση πολλών υπολογισμών σε μικρό dataset θα επέτρεπε τη χρήση μεγαλύτερων τύπων δεδομένων όπως δουβλε χωρίς κάποια σημαντική διαφορά.

Συνεπώς θεωρώ ότι με αρκετό πειραματισμό μεταξύ μικρών και μεγάλων τύπων δεδομένων καθώς επίσης και με βάση το μέγεθος του dataset του Movielens που χρησιμοποιείται κάθε φορά είτε 100.000 βαθμολογιών, είτε 1.000.000 βαθμολογιών, είτε 10.000.000 βαθμολογιών, είτε 20.000.000 βαθμολογιών μπορεί να προκύψει βελτίωση στον χρόνο υλοποιήσης των προγραμμάτων λαμβάνοντας υπόψην και την αρχιτεκτονική του συστήματος κάθε φορά x86 ή x64.

7.4 Μελλοντικές επεκτάσεις

Οι μελλοντικές επεκτάσεις που αναφέρονται από κάτω αφορούν το σειριακό και το παράλληλο πρόγραμμα που υλοποιήθηκαν σε Java και όχι στον αρχικό κώδικα που δόθηκε από τον ιστότοπο υλοποιημένο σε C++. Καθώς ο τελευταίος παραμετροποιήθηκε από μέρος μου έτσι ώστε να δέχεται τα dataset του MovieLens σε C++.

Μελλοντικά, θα μπορούσαν να επεκταθούν τα δύο προγράμματα έτσι ώστε να μπορούν να διαβάζουν τα dataset του Netflix καθώς διαφέρει ο τρόπος που είναι δομημένη η πληροφορία συγκριτικά με το τρόπο που είναι διαθέσιμη στο dataset του MovieLens. Στο dataset του MovieLens το αρχείο u.data περιέχει όλες τις ταυτότητες των χρηστών, όλες τις ταυτότητες των ταινιών, όλες τις βαθμολογίες που έχουν δοθεί από τους χρήστες σε ταινίες μαζί με το timestamp τους. Ουσιαστικά το dataset του MovieLens διαθέτει όλη την αξιποιήσιμη πληροφορία συγκεντρωμένη σε ένα αρχείο. Αντίθετα στη περίπτωση του Netflix η πληροφορία του trading dataset έχει κατανεμηθεί σε 17.700 ξεχωριστά αρχεία. Συνεπώς, έχοντας αυτό σαν δεδομένο διαφέρει ο αριθμός των αρχείων που πρέπει να διαβαστούν, επιπρόσθετα υπάρχουν και άλλες διαφορές όπως θα δούμε στη συνέχεια.

Μια άλλη διαφορά είναι και ο τρόπος που διανέμονται οι πληροφορίες στα αρχεία. Στη περίπτωση του MovieLens τα δεδομένα δίνονται σε μια λίστα από γραμμές στις οποίες τα πεδία είναι χωρισμένες με tab. Τα πεδία σε κάθε γραμμή δίνονται με την ακόλουθη σειρά, η ταυτότητα ενός χρήστη, ύστερα η ταυτότητα της ταινίας, έπειτα η βαθμολογία που έχει δώσει ο καθορισμένος χρήστης στην συγκεκριμένη ταινία και τελικά στο τέλος κάθε γραμμής ακολουθεί μια χρονοσφραγίδα. Ενώ στη περίπτωση του Netflix στο φάκελο training_set περιέχονται 17.700 αρχεία, κάθε αρχείο αντιπροσωπεύει μία από τις ταινίες και σε κάθε αρχείο η πληροφορία είναι παρουσιασμένη με τον εξής τρόπο, στη πρώτη γραμμή κάθε αρχείου υπάρχει η ταυτότητα της ταινίας και από κάτω ακολουθεί μια λίστα από γραμμές στις οποίες τα δεδομένα έχουν την εξής σειρά, η ταυτότητα του χρήστη, η βαθμολογία που έχει δώσει κάθε χρήστης στη ταινία και τέλος η ημερομηνία που δόθηκε η βαθμολογία. Επιπλέον τα πεδία δεν είναι χωρισμένα με tab αλλά με κόμμα.

Βιβλιογραφία

[1] Wikipedia: Recommender system, https://en.wikipedia.org/wiki/Recommender_system

[2] Wikipedia: Collaborative filtering, https://en.wikipedia.org/wiki/Collaborative_filtering.

[3] Wikipedia: Singular value decomposition, https://en.wikipedia.org/wiki/Singular_value_decomposition

[4] Timely Development, Netflix Prize, 2007, http://www.timelydevelopment.com/demos/NetflixPrize.aspx

[5] Simon Funk. [Netflix Update: Try This at Home]. Journal, Monday, December 11, 2006, http://sifter.org/ simon/journal/20061211.html

[6] Wikipedia: Data set, https://en.wikipedia.org/wiki/Data set

[7] Wikipedia: MovieLens, https://en.wikipedia.org/wiki/MovieLens

[8] Wikipedia: MovieLens, http://grouplens.org/datasets/movielens/

[9] Apache Spark, Lightning-fast cluster computing, http://spark.apache.org/docs/latest/

[10] Gediminas Adomavicius, Alexander Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, Paper, IEEE Transactions on knowledge and data engineering, vol. 17, No. 6, June 2005.

[11] John S. Breese, David Heckerman, Carl Kadie, Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Microsoft Research, October 1998.

[12] ήμητρα Παρασκευοπούλου, Κατηγοριοποίηση κειμένων με τη βοήθεια ταξινομητών, ιπλωαματική Εργασία, http://vivliothmmy.ee.auth.gr/346/1/

[13] [SVD Tutorial], Article, http://alias-i.com/lingpipe/demos/tutorial/svd/read-me.html

[14] HowtoForge, Setting up an NFS server and client on centOS 6.3, https://www.howtoforge.com/setting-up-an-nfs-server-and-client-on-centos-6.3

[15] Spark, Spark Standalone Mode, Documentation, Programming Guide, http://spark.apache.org/docs/latest/spark-standalone.html

[16] Spark, Spark Programming Guide, Documentation, Programming Guide, http://spark.apache.org/docs/latest/programming-guide.html

[17] Spark, API Docs, Java, http://spark.apache.org/docs/latest/api/java/index.html

[18] Gliffy, cloud-based diagramming software, https://www.gliffy.com/

[19] Matei Zaharia, Parallel programming with Spark, Presentation, UC Berkeley, http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/matei-zaharia-part-1-amp-camp-2012-spark-intro.pdf

[20] ShareLaTex, online LaTeX editor, https://www.sharelatex.com/

[21] Academic Torrents, Netflix Prize Data Set, http://academictorrents.com/details/9b13183dc4d60676b773c9e2cd6de5e5542cee9a

[22] Grouplens, MovieLens datasets, http://grouplens.org/datasets/movielens/

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

[24] Stack Overflow, Differentiate driver code and work code in Apache Spark, December 2015, http://stackoverflow.com/questions/33339200/differentiate-driver-code-and-work-code-in-apache-spark

[25] GitBook, Mastering Apache Spark, TaskScheduler, https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-taskscheduler.html

[26] Safari, Chapter 5. Loading and Saving Your Data, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch05.html

[27] Cloudera, How-to: Tune your apache spark jobs 1, http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-1/

[28] Cloudera, How-to: Tune your apache spark jobs 2, http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

[29] Spark Apache, Submitting Applications, http://spark.apache.org/docs/latest/submitting-applications.html

[30] Khoa's IT blog, understand the shuffle component in spark-core, April 4, 2015, https://trongkhoanguyenblog.wordpress.com/

[31] Apache Spark, Spark Configuration, http://spark.apache.org/docs/latest/configuration.html

[32] Apache Spark, Building Spark, http://spark.apache.org/docs/latest/building-spark.html

[33] Oracle, The Java™ Tutorials, Local Classes, https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html

[34] Oracle, The Java™ Tutorials, Lambda Expressions, https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

[35] Safari, Chapter 3. Programming with RDDs, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch03.html

[36] Oracle, The Java™ Tutorials, Anonymous Classes, https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html

[37] Oracle, The Java™ Tutorials, Nested Classes, https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

[38] Safari, Chapter 1. Introduction to Data Analysis with Spark, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch01.html

[39] Safari, Chapter 2, Downloading Spark and Getting Started, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch02.html

[40] Safari, Chapter Index, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ix01.html

[41] Safari, Chapter 8. Tuning and Debugging Spark, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch08.html

[42] Taming The Elephant, Using KryoSerializer in SPARK. Friday, February 13, 2015, http://lordjoesoftware.blogspot.gr/2015/02/using-kryoserializer-in-spark.html

[43] Spark Summit 2015, Session, IndexedRDD: Efficient Fine-Grained Updates for RDDs, Monday, June 15, https://spark-summit.org/2015/events/indexedrdd-efficient-fine-grained-updates-for-rdds/

[44] Safari, Chapter 4. Working with Key/Value Pairs, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch04.html

[45] Safari, Chapter 6. Advanced Spark Programming, https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch06.html

Παραρτήματα

Ακολουθιακό Πρόγραμμα C++ με MovieLens datasets

Και τα τρία προγράμματα που ακολουθούν εμπεριέχουν επεξηγηματικά σχόλια.

Ακολουθεί από κάτω η μετατροπή του αρχικού κώδικα σε C++ έτσι ώστε να δέχεται ως είσοδο τα dataset του MovieLens.

```
//=============================================================================
//
// SVD Sample Code
//
// Copyright (C) 2007 Timely Development (www.timelydevelopment.com)
//
// Special thanks to Simon Funk and others from the Netflix Prize contest
// for providing pseudo-code and tuning hints.
//
// Feel free to use this code as you wish as long as you include
// these notices and attribution.
//
// Also, if you have alternative types of algorithms for accomplishing
// the same goal and would like to contribute, please share them as well :)
//
// STANDARD DISCLAIMER:
//
// - THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY
// - OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT
// - LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR
// - FITNESS FOR A PARTICULAR PURPOSE.
//
```

```
//==========================================================================

#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include  <tchar.h>
#include <map>
using namespace std;

//===================================================================
//
// Constants and Type Declarations
//
//===================================================================

#define TRAINING_FILE    "C:\\MovieLens\\u.data"
#define TESTING_FILE "C:\\MovieLens\\u1.test"
#define PREDICTION_FILE "C:\\MovieLens\\u1.predictions"

#define  MAX_RATINGS        100000      //  Ratings  in  entire  training  set  (+1)
#define  MAX_CUSTOMERS      943         // Customers  in  the  entire  training  set  (+1)
#define  MAX_MOVIES         1682          // Movies  in  the  entire  training  set  (+1)
#define MAX_FEATURES        64              //  Number  of  features  to use
#define MIN_EPOCHS          120             // Minimum  number  of  epochs  per  feature
#define  MAX_EPOCHS         200             //  Max  epochs  per  feature

#define MIN_IMPROVEMENT 0.0001              // Minimum  improvement required to continue current  feature
#define INIT                0.1             // Initialization  value  for  features
#define LRATE               0.001           // Learning rate  parameter
#define K                   0.015           // Regularization parameter used to minimize   over-fitting
```

```
typedef unsigned char BYTE;

struct Movie
{
public:
	int		RatingCount = 0;
	int		RatingSum = 0;

	double RatingAvg()
	{
		return RatingSum / (1.0 * RatingCount);
	}

	double PseudoAvg()
	{
		return (3.23 * 25 + RatingSum) / (25.0 + RatingCount);
	}

};

struct Customer
{
	int		RatingCount = 0;
	int		RatingSum = 0;
};

struct Data
{
	int		CustId;
	short		MovieId;
```

```
    BYTE        Rating;
    double      Cache = 0.0;
};

class Engine
{
private:
int       m_nRatingCount;    // Current number of loaded ratings
Data      m_aRatings[MAX_RATINGS]; // Array of ratings data
Movie     m_aMovies[MAX_MOVIES+1]; // Array of movie metrics
Customer  m_aCustomers[MAX_CUSTOMERS+1]; // Array of customer metrics
double    m_aMovieFeatures[MAX_FEATURES][MAX_MOVIES+1]; // Array of features by movie
double    m_aCustFeatures[MAX_FEATURES][MAX_CUSTOMERS+1]; // Array of features by customer

inline double PredictRating(short movieId, int custId, int feature, double cache, bool bTrailing=true);
inline double PredictRating(short movieId, int custId);

public:
    Engine(void);
    ~Engine(void) { };

    void        CalcMetrics();
    void        CalcFeatures();
    void        LoadHistory();
    void        ProcessTest();
    void        ProcessFile(const char* pwzFile);
};


//===================================================================
//
```

```
// Program Main
//
//==================================================================
int _tmain(int argc, _TCHAR*  argv[])
{
	Engine*  engine  =  new  Engine();

	engine->LoadHistory();
	engine->CalcFeatures();
	engine->ProcessTest();

	wprintf(L"\nDone\n");
	getchar();

	return 0;
}


//==================================================================
//
// Engine Class
//
//==================================================================

//------------------------------------------------------------------
// Initialization
//------------------------------------------------------------------

Engine::Engine(void)
{
	m_nRatingCount = 0;
```

```
        for (int f = 0; f<MAX_FEATURES; f++)
        {
                for (int i = 1; i<MAX_MOVIES+1; i++) m_aMovieFeatures[f][i] = INIT;
                for (int i = 1; i<MAX_CUSTOMERS+1; i++) m_aCustFeatures[f][i] = INIT;
        }
}

//-------------------------------------------------------------------
// Calculations - This Paragraph contains all of the relevant code
//-------------------------------------------------------------------

//
// CalcFeatures
// - Iteratively train each feature on the entire data set
// - Once sufficient progress has been made, move on
//
void Engine::CalcFeatures()
{
        int f, e, i, custId, cnt = 0;
        Data* rating;
        double err, p, sq, rmse_last = 0.0, rmse = 2.0;
        short movieId;
        double cf, mf;

        for (f = 0; f<MAX_FEATURES; f++)
        {
                wprintf(L"\n--- Calculating feature: %d ---\n", f);

                // Keep looping until you have passed a minimum number
                // of epochs or have stopped making significant progress
```

```
for (e = 0; (e < MIN_EPOCHS) || (rmse <= rmse_last - MIN_IMPROVEMENT); e++)
{
        cnt++;
        sq = 0;
        rmse_last = rmse;

        for (i = 0; i<m_nRatingCount; i++)
        {
                rating = m_aRatings + i;
                movieId = rating->MovieId;
                custId = rating->CustId;

                // Predict rating and calc error
                p = PredictRating(movieId, custId, f, rating->Cache, true);
                err = (1.0 * rating->Rating - p);
                sq += err*err;

                // Cache off old feature values
                cf = m_aCustFeatures[f][custId];
                mf = m_aMovieFeatures[f][movieId];

                // Cross-train the features
                m_aCustFeatures[f][custId] += (LRATE * (err * mf - K * cf));
                m_aMovieFeatures[f][movieId] += (LRATE * (err * cf - K * mf));
        }

        rmse = sqrt(sq / m_nRatingCount);

        //wprintf(L"        <set x='%d' y='%f' />\n", cnt, rmse);
}
```

```
            // Cache  off  old predictions
            for (i = 0; i<m_nRatingCount; i++)
            {
               rating = m_aRatings +   i;
               rating->Cache = PredictRating(rating->MovieId, rating->CustId, f, rating->Cache, false);
            }
        }
}

//
// PredictRating
// - During training there is no need to loop through all of the features
// - Use a cache for the leading features and do a quick calculation for the trailing
// - The trailing can be optionally removed when calculating a new cache value
//
double Engine::PredictRating(short movieId, int custId, int feature, double cache, bool   bTrailing)
{
        // Get cached value for old features or default to an   average
        double sum = (cache > 0) ? cache : 1; //m_aMovies[movieId].PseudoAvg;

        // Add contribution of current feature
        sum  +=  m_aMovieFeatures[feature][movieId]  *  m_aCustFeatures[feature][custId];
        if (sum > 5) sum =    5;
        if (sum  <  1) sum  = 1;

        // Add up trailing defaults values
        if (bTrailing)
        {
                sum += (MAX_FEATURES - feature - 1) * (INIT * INIT);
                if  (sum  >  5)  sum  =  5;
                if (sum  <  1) sum  = 1;
```

```
    }

    return sum;
}

//
// PredictRating
// - This version is used for calculating the final results
// - It loops through the entire list of finished features
//
double Engine::PredictRating(short movieId, int custId)
{
    double sum = 1; //m_aMovies[movieId].PseudoAvg;

    for (int f = 0; f<MAX_FEATURES; f++)
    {
        sum += m_aMovieFeatures[f][movieId] * m_aCustFeatures[f][custId];
        if (sum > 5) sum = 5;
        if (sum < 1) sum = 1;
    }

    return sum;
}

//-------------------------------------------------------------------
// Data Loading / Saving
//-------------------------------------------------------------------

//
// LoadHistory
// - Loop through all of the files in the training directory
```

```
//
void Engine::LoadHistory()
{

	this->ProcessFile(TRAINING_FILE);
}


void  Engine::ProcessFile(const  char*  pwzFile)
{
	int custId, movieId, rating, items;
	long tmp;
	FILE *in;

	// open input file
	if  ((in  =  fopen((const  char*)pwzFile,  "r"))  ==  NULL)		return;

	// repeat as long as the EOF has not been reached
	while  (!feof(in))
	{
		// read TAB  delimited line
		items  =  fscanf(in,  "%d\t%d\t%d\t%d\n",  &custId,  &movieId,  &rating,  &tmp);
		// assign tokens read from input file to the corresponding m_aRatings[] struct
		m_aRatings[m_nRatingCount].MovieId = (short)movieId;
		m_aRatings[m_nRatingCount].CustId  =  custId;
		m_aRatings[m_nRatingCount].Rating   =   (BYTE)rating;
		// update Movie and Customer statistics
		m_aMovies[movieId].RatingCount++;
		m_aCustomers[custId].RatingCount++;
		m_aMovies[movieId].RatingSum += rating;
		m_aCustomers[custId].RatingSum  +=  rating;
```

```
                // increment ratings count
                m_nRatingCount++;
        }
        // close inputfile
        fclose(in);
}

//
// ProcessTest
// - Load a sample set in the following format

void Engine::ProcessTest()
{
        FILE *in, *out;
        long tmp;
        int custId = 0;
        short movieId = 0;
        unsigned char rating = 0;
        double predictrating = 0.0, diff;
        double sum = 0.0; // the sum of all differences
        long cnt = 0, // the number of predictions
                items;

        // open input file for read
        if ((in = fopen((const char*)TESTING_FILE, "r")) == NULL)        return;
        // open output file for write
        if ((out = fopen((const char*)PREDICTION_FILE, "w")) == NULL)        return;

        // repeat as long as the EOF of the input file has not been reached
        while (!feof(in))
        {
```

```
            // read TAB delimited line
            items = fscanf(in, "%d\t%d\t%d\t%d\n", &custId, &movieId, &rating, &tmp);
            // calculate the prediction
            predictrating = PredictRating(movieId, custId);
            // calculate prediction statistics
            diff = abs(predictrating - rating);
            sum += diff;
            cnt++;
            // write the TAB delimited prediction line
            items = fprintf(out, "%d\t%d\t%d\t%.3f\t%.3f\n", custId, movieId, rating,
                predictrating,   diff);
        }
  // write the summary line
  items = fprintf(out, "\n----\nNumber of predictions : %d\nAvg Abs(diff) : %0.6f\n", cnt, sum / cnt);
  // close files
  fclose(in);
  fclose(out);
}
```

Ακολουθιακό Πρόγραμμα Java με MovieLens datasets

Ακολουθεί από κάτω η υλοποίηση του σειριακού κώδικα σε γλώσσα προγραμματισμού Java χρησιμοποιώντας ως είσοδο το MovieLens Dataset των 100.000 βαθμολογιών.

```
package SVDMovieLensJava;

public class Customer {
	//-------------------------------DATA MEMBERS ------------
	public int RatingCount = 0;
	public int RatingSum = 0;
}



package SVDMovieLensJava;

public class Data {
//-------------------------------DATA MEMBERS ------------
	public int CustId;
	public short MovieId;
	public byte Rating;
	public double Cache = 0.0;
}



package SVDMovieLensJava;

public class Movie {
	//-------------------------------DATA MEMBERS ------------
```

```
    public int RatingCount = 0;
    public  int  RatingSum  = 0;

    //------------------------------  METHODS  ------------
    public double RatingAvg()
    {
        return RatingSum / (1.0 * RatingCount);
    }
    public  double PseudoAvg()
    {
        return  (3.23  *  25  +  RatingSum)  / (25.0  +  RatingCount);
    }
}



package  SVDMovieLensJava;

import java.io.File;
import java.io.FilenameFilter;


public class myFileNameFilter implements FilenameFilter {
    String   filter;
    public myFileNameFilter(String value)
    {
        this.filter   =  value;
    }

    @Override
    public boolean accept(File dir, String name)    {
```

```java
            return name.endsWith(this.filter);
        }
}
```

```java
package SVDMovieLensJava;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.Duration;
import java.util.Scanner;
import java.time.LocalTime;

public class Engine {

    //--------------------------------- Data Members --------------------------
private final String TRAINING_FILE = "/nfs/MovieLens/u.data";
private final String TESTING_FILE = "/nfs/MovieLens/u1.test";
private final String PREDICTIONS_FILE = "/nfs/MovieLens/u1.predictions";

private final int MAX_RATINGS = 100000; // Ratings in entire training set (+1)
private final int MAX_CUSTOMERS = 943; // Customers in the entire training set (+1)
private final int MAX_MOVIES = 1682;    // Movies in the entire training set (+1)
private final int MAX_FEATURES = 64;    // Number of features to use
private final int MIN_EPOCHS = 120;     // Minimum number of epochs per feature
```

```
private final int MAX_EPOCHS = 200;          // Max epochs per feature

private final double MIN_IMPROVEMENT = 0.0001;//Min improvement required to continue current feature
private final double INIT   = 0.1;    //  Initialization value for features
private final double LRATE = 0.001; // Learning rate  parameter
private final double K = 0.015; // Regularization parameter used to minimize over-fitting

private int m_nRatingCount;     // Current number of loaded ratings
private Data[] m_aRatings = new Data[MAX_RATINGS];  // Array of ratings data
private Movie[] m_aMovies = new Movie[MAX_MOVIES + 1];  // Array of movie metrics
private Customer[] m_aCustomers = new Customer[MAX_CUSTOMERS + 1];// Array of customer metrics
private double[][] m_aMovieFeatures = new double[MAX_FEATURES][MAX_MOVIES + 1];//Array of f by mov.
private double[][] m_aCustFeatures = new double[MAX_FEATURES][MAX_CUSTOMERS + 1];//Array of f by cust.

    //---------------------------------------------Constructor--------------
    public Engine() {
        int f, i;
        this.m_nRatingCount = 0;

        for (f = 0; f < this.MAX_FEATURES; f++) {
            for (i = 1; i < this.MAX_MOVIES+1; i++) {
                this.m_aMovieFeatures[f][i] = this.INIT;
            }
            for (i = 1; i < this.MAX_CUSTOMERS+1; i++) {
                this.m_aCustFeatures[f][i] = this.INIT;
            }
        }
    }

    //--------------------------Member Functions of Engine Class---------------
    private void CheckUpdateMovies(short movieId, byte rating) {
```

```
    // check if movie ID does not exist
    if (this.m_aMovies[movieId] == null) {
        this.m_aMovies[movieId] = new Movie();
    }
    // update movie record
    this.m_aMovies[movieId].RatingCount++;
    this.m_aMovies[movieId].RatingSum += rating;
}

private void CheckUpdateCustomers(short custId, byte rating) {
    // check if customer ID does not exist
    if (this.m_aCustomers[custId] == null) {
        this.m_aCustomers[custId] = new Customer();
    }
    // update customer record
    this.m_aCustomers[custId].RatingCount++;
    this.m_aCustomers[custId].RatingSum += rating;
}

//-------------------------ProcessFile-----------------------------------
private void ProcessFile(String filename) throws IOException {
    BufferedReader inputStream = null;
    Scanner sc = null;
    short custId = 0, movieId = 0;
    byte rating = 0;
    //============try-catch-finally=====================================
    try {
        String line;
        Data data;
        inputStream = new BufferedReader(new FileReader(filename));
        while ((line = inputStream.readLine()) != null) {
```

```
            // read a customer ID, movie ID and rating
            sc = new Scanner(line);
            sc.useDelimiter("\t");
            custId = Short.parseShort(sc.next());
            movieId = Short.parseShort(sc.next());
            rating   =  Byte.parseByte(sc.next());
            //--Update-Ratings-array-------------------------------
            this.m_aRatings[this.m_nRatingCount] = new  Data();
            data = this.m_aRatings[this.m_nRatingCount];
            data.MovieId  = movieId;
            data.CustId = custId;
            data.Rating   =  rating;
            //--Update--Movie--Statistics--------------------------------
            CheckUpdateMovies(movieId,  rating);
            //--Update--Customer--Statistics--------------------------------
            CheckUpdateCustomers(custId, rating);
            // increment ratings count
            this.m_nRatingCount++;
            /*System.out.printf("movieId = %d, custId = %d, rating = %d, ratings = %d\n",
                        movieId,   custId,   rating,  this.m_nRatingCount);*/
        }
    } catch (FileNotFoundException e) {
        System.out.printf("<Engine.ProcessFile()>Error opening  file %s\n", movieId);
    }  finally   {
        if (inputStream != null) {
            inputStream.close();
        }
        if (sc  !=  null) {
            sc.close();
        }
    }
```

```
    //=================================================================
}

//-------------------------LoadHistory-----------------------------------
public void LoadHistory() throws IOException {
    this.ProcessFile(this.TRAINING_FILE);
}

//
// PredictRating
// - During training there is no need to loop through all of the features
// - Use a cache for the leading features and do a quick calculation for the trailing
// - The trailing can be optionally removed when calculating a new cache value
//
double PredictRating(short movieId, int custId, int feature, double cache, boolean bTrailing) {
    // Get cached value for old features or default to an average
    double sum = (cache > 0) ? cache : 1; //m_aMovies[movieId].PseudoAvg;

    // Add contribution of current feature
    sum += this.m_aMovieFeatures[feature][movieId] * m_aCustFeatures[feature][custId];
    if (sum > 5) {
        sum = 5;
    }
    if (sum < 1) {
        sum = 1;
    }

    // Add up trailing defaults values
    if (bTrailing) {
        sum += (MAX_FEATURES - feature - 1) * (INIT * INIT);
        if (sum > 5) {
```

```
            sum  = 5;
        }
        if (sum < 1) {
            sum = 1;
        }
    }
    return sum;
}

//
// CalcFeatures
// - Iteratively train each  feature on  the entire data    set
// - Once sufficient progress has been made, move    on
//
void  CalcFeatures()  {
    int f, e, i, custId, cnt = 0;
    Data rating;
    double err, p, sq, rmse_last = 0.0, rmse = 2.0;
    short movieId;
    double  cf, mf;

    for (f = 0; f < this.MAX_FEATURES; f++) {
        System.out.printf("\n--- Calculating feature: %d ---\n", f);
        // Keep  looping until you have passed a minimum    number
        // of epochs or have stopped making significant   progress
        for (e = 0; (e < this.MIN_EPOCHS) ||  (rmse <= rmse_last - this.MIN_IMPROVEMENT); e++) {
            cnt++;
            sq  = 0;
            rmse_last  =  rmse;
            for (i  = 0; i  < this.m_nRatingCount; i++) {
                rating =  this.m_aRatings[i];
```

```
                movieId = rating.MovieId;
                custId  =  rating.CustId;

                // Predict rating and calc error
                p = PredictRating(movieId, custId, f, rating.Cache, true);
                err = (1.0 * rating.Rating - p);
                sq += err * err;

                // Cache off old feature values
                cf  =  m_aCustFeatures[f][custId];
                mf  =  m_aMovieFeatures[f][movieId];

                // Cross-train the features
                m_aCustFeatures[f][custId] += (LRATE * (err * mf - K * cf));
                m_aMovieFeatures[f][movieId] += (LRATE * (err * cf - K * mf));
            }

            rmse = Math.sqrt(sq / m_nRatingCount);

            /*System.out.printf("        <set x='%d' y='%f' />\n", cnt, rmse);*/
        }
        // Cache off old predictions
        for (i = 0; i < this.m_nRatingCount; i++) {
            rating = m_aRatings[i];
            rating.Cache = PredictRating(rating.MovieId, rating.CustId, f, rating.Cache, false);
        }
    }
}

//
// PredictRating
```

```
// - This version is used for calculating the final results
// - It loops through the entire list of finished features
//
double PredictRating(short movieId, int custId) {
    double sum = 1; //m_aMovies[movieId].PseudoAvg;

    for (int f = 0; f < this.MAX_FEATURES; f++) {
        sum += this.m_aMovieFeatures[f][movieId] * this.m_aCustFeatures[f][custId];
        if (sum > 5) {
            sum = 5;
        }
        if (sum < 1) {
            sum = 1;
        }
    }
    return sum;
}

//
// ProcessTest
// - Load a sample set in the following format
//
void ProcessTest() throws IOException {
    BufferedReader inputStream = null;
    PrintWriter out = null;
    Scanner sc = null;
    short custId = 0, movieId = 0;
    byte rating = 0;
    double predictrating = 0.0, diff;
    double sum = 0.0;
    int cnt = 0;
```

```
//=============try-catch-finally=======================================
try {
    String line;
    Data data;
    inputStream = new BufferedReader(new FileReader(this.TESTING_FILE));
    out = new PrintWriter(new BufferedWriter(new FileWriter(this.PREDICTIONS_FILE)));
    while ((line = inputStream.readLine()) != null) {
        // read a customer ID, movie ID and rating
        sc = new Scanner(line);
        sc.useDelimiter("\t");
        custId = Short.parseShort(sc.next());
        movieId = Short.parseShort(sc.next());
        rating = Byte.parseByte(sc.next());
        predictrating = PredictRating(movieId, custId);
        diff = Math.abs(predictrating - rating);
        sum += diff;
        cnt++;
        out.format("%d\t%d\t%d\t%.3f\t%.3f\n", custId, movieId, rating,
                predictrating, diff);
        /* System.out.printf("movieId = %d, custId = %d\n",
                movieId, custId);*/
    }
    out.format("\n------\nNumber of predictions : %d\nAvg Abs(diff) : %f\n", cnt, sum / cnt);
} catch (FileNotFoundException e) {
    System.out.printf("<Engine.ProcessFile()>Error opening file %s\n", movieId);
} finally {
    if (inputStream != null) {
        inputStream.close();
    }
    if (out != null) {
        out.close();
```

```
            }
            if (sc  !=  null) {
                sc.close();
            }
        }
    }

    //------------------------------Main()-----------------------------------
public static void main(String[] args) throws IOException {
 LocalTime t1, t2, t3, t4,    t5;

 t1  = LocalTime.now();
 Engine engine = new  Engine();
 t2   =  LocalTime.now();
 System.out.printf("engine construction duration equals %d s\n",Duration.between(t1, t2).getSeconds());
 engine.LoadHistory();
 t3  = LocalTime.now();
 System.out.printf("load history duration equals %d s\n",Duration.between(t2, t3).getSeconds());
 engine.CalcFeatures();
 t4  = LocalTime.now();
 System.out.printf("calculation feature duration equals %d s\n",Duration.between(t3, t4).getSeconds());
 engine.ProcessTest();
 t5  = LocalTime.now();
 System.out.printf("processing test duration equals %d s\n",Duration.between(t4, t5).getSeconds());
 System.out.println("\nDone\n");
}
    //-------------------------------------------------------------------------
}
```

Παράλληλο Πρόγραμμα **Java** στο **Apache Spark** με **MovieLens datasets**

Παρακάτω παρουσιάζεται η υλοποίηση του παράλληλου κώδικα στη διεπαφή προγραμματισμού εφαρμογών Apache Spark χρησιμοποιώντας την υποστηριζόμενη γλώσσα προγραμματισμού Java

```
package edu.berkeley.svdmovielens;

import java.io.Serializable;

public class Customer implements Serializable {
    //------------------------------DATA MEMBERS ------------
    public int RatingCount = 0;
    public int RatingSum = 0;

    public Customer() {}

    public Customer(int RatingCount, int RatingSum) {
        this.RatingCount = RatingCount; this.RatingSum
        = RatingSum;
    }

    public void setRatingCount(int RatingCount) {
        this.RatingCount = RatingCount;
    }

    public void setRatingSum(int RatingSum) {
        this.RatingSum = RatingSum;
    }
```

```java
	public int getRatingCount() {
		return RatingCount;
	}

	public int getRatingSum() {
		return RatingSum;
	}

	@Override
	public String toString() {
		return "Customer [ratingCount=" + RatingCount + ", ratingSum=" + RatingSum + "]";
	}
}
```

```java
package edu.berkeley.svdmovielens;

import java.io.Serializable;

public class Movie implements Serializable {
	//-------------------------------DATA MEMBERS ------------
	public int RatingCount = 0;
	public int RatingSum = 0;

	public Movie() {}

	public Movie(int RatingCount, int RatingSum) {
		this.RatingCount = RatingCount;
		this.RatingSum = RatingSum;
	}
```

```java
public int getRatingCount() {
    return RatingCount;
}

public void setRatingCount(int RatingCount) {
    this.RatingCount  =  RatingCount;
}

public int getRatingSum() {
    return RatingSum;
}

public void setRatingSum(int RatingSum) {
    this.RatingSum = RatingSum;
}

//------------------------------  METHODS  ------------
public double RatingAvg()
{
    return RatingSum / (1.0 * RatingCount);
}
public  double PseudoAvg()
{
    return  (3.23  *  25  +  RatingSum)  / (25.0  +  RatingCount);
}

@Override
public  String  toString()  {
    return "Movie [ratingCount=" + RatingCount + ", ratingSum=" + RatingSum + "]";
}
```

```
}



package edu.berkeley.svdmovielens;

import  java.io.Serializable;

public  class  SparkData  implements  Serializable {
//------------------------------DATA  MEMBERS  ------------
    public int CustId;
    public int MovieId;
    public int Rating;

    public  SparkData() {}

    public SparkData(int CustId, int MovieId, int Rating) {
        this.CustId  = CustId;
        this.MovieId = MovieId;
        this.Rating   =   Rating;
    }

    public int getCustId() {
        return CustId;
    }

    public void setCustId(int CustId) {
        this.CustId  = CustId;
    }

    public int getMovieId()  {
```

```
        return MovieId;
    }

    public void setMovieId(int MovieId) {
        this.MovieId  =  MovieId;
    }

    public int getRating() {
        return Rating;
    }

    public void setRating(int Rating) {
        this.Rating  = Rating;
    }

    @Override
    public  String  toString()  {
        return "SparkData [custId=" + CustId + ", MovieId=" + MovieId + ", Rating=" + Rating + "]";
    }
}
```

```
package edu.berkeley.svdmovielens;

public class TrainingData extends SparkData    {

    public  double  Cache  = 0;

    public TrainingData(int CustId, int MovieId, int Rating) {
        super(CustId, MovieId, Rating);
```

```java
    }
    public double getCache() {
        return  Cache;
    }

    public void setCache(double Cache) {
        this.Cache  =  Cache;
    }

    @Override
    public  String  toString()  {
        return  "TrainingData  [custId="  +  CustId  +  ", MovieId="  +  MovieId +
                    + ", Rating=" + Rating + ", Cache=" + Cache + "]";
    }
}



package edu.berkeley.svdmovielens;

public  class  TestingData  extends  TrainingData {

    // data members
    public double  PredictRating;

    // constructor
    public TestingData(int CustId, int MovieId, int Rating, double PredictRating) {
        super(CustId, MovieId, Rating);
        this.PredictRating   =   PredictRating;
    }
```

```java
    public TestingData(TrainingData data, double PredictRating) {
        super(data.CustId, data.MovieId, data.Rating);
        this.PredictRating  =  PredictRating;
    }

    // methods
    public double diff() {
        return Math.abs(Rating - PredictRating);
    }

    @Override
    public String toString() {
        return CustId + "\t" + MovieId + "\t" + Rating + "\t" + PredictRating + "\t" + this.diff();
    }
}
```

```java
package edu.berkeley.svdmovielens;

import com.esotericsoftware.kryo.Kryo;
import com.esotericsoftware.kryo.serializers.FieldSerializer;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Serializable;
import java.time.Duration;
```

```java
import java.util.Scanner;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.SparkConf;
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.api.java.JavaPairRDD;
import scala.Tuple2;
import org.apache.spark.serializer.KryoRegistrator;
import org.apache.spark.Accumulator;
import org.apache.spark.api.java.function.PairFunction;

public class SVDMovieLensSparkJava implements Serializable {

// ------------------------ Serial version Members ---------------------------
private final int MAX_RATINGS = 100000; // Ratings in entire training set (+1)
private final int MAX_CUSTOMERS = 943; // Customers in the entire training set (+1)
private final int MAX_MOVIES = 1682;      // Movies in the entire training set (+1)
private final int MAX_FEATURES = 64;     // Number of features to use
private final int MIN_EPOCHS = 120;        // Minimum number of epochs per feature
private final int MAX_EPOCHS = 200;       // Max epochs per feature
private final double MIN_IMPROVEMENT = 0.0001;// Minimum improvement required to continue current f.
private final double INIT = 0.1;     // Initialization value for features
private final double LRATE = 0.001; // Learning rate parameter
private final double K = 0.015; // Regularization parameter used to minimize over-fitting
```

```
private int m_nRatingCount;      // Current number of loaded ratings
private TrainingData[] m_aRatings = new TrainingData[MAX_RATINGS];  // Array of ratings data
private Movie[] m_aMovies = new Movie[MAX_MOVIES + 1];  // Array of movie metrics
private Customer[] m_aCustomers = new Customer[MAX_CUSTOMERS + 1];// Array of customer metrics
private double[][] m_aMovieFeatures = new double[MAX_FEATURES][MAX_MOVIES + 1];//2D of features by mov.
private double[][] m_aCustFeatures = new double[MAX_FEATURES][MAX_CUSTOMERS + 1];//2D of features by cu
//-------------------------------- Spark version Members --------------------------
private final transient String SparkAppName = "SVD MovieLens - Apache Spark";
private transient SparkConf conf;
private transient JavaSparkContext sc;
private final Broadcast<String> TRAINING_FILE_BRDCST;
private final Broadcast<String> TESTING_FILE_BRDCST;
private final Broadcast<String> PREDICTIONS_FILE_BRDCST;
private final Broadcast<Integer> MAX_RATINGS_BRDCST; // Ratings in entire training set (+1)
private final Broadcast<Integer> MAX_CUSTOMERS_BRDCST; // Customers in the entire training set (+1)
private final Broadcast<Integer> MAX_MOVIES_BRDCST;   // Movies in the entire training set (+1)
private final Broadcast<Integer> MAX_FEATURES_BRDCST;  // Number of features to use
private final Broadcast<Integer> MIN_EPOCHS_BRDCST;       // Minimum number of epochs per feature
private final Broadcast<Integer> MAX_EPOCHS_BRDCST;     // Max epochs per feature
private final Broadcast<Double> MIN_IMPROVEMENT_BRDCST;// Minimum improvement requir. to cont. current
private final Broadcast<Double> INIT_BRDCST;    // Initialization value for features
private final Broadcast<Double> LRATE_BRDCST; // Learning rate parameter
private final Broadcast<Double> K_BRDCST; // Regularization parameter used to minimize over-fitting
private Broadcast<Long> m_nRatingCount_BRDCST;     // Current number of loaded ratings
private JavaPairRDD<Integer, TrainingData> m_aTrainingRatings_PairRDD;  // RDD of ratings data
private JavaPairRDD<Integer, Movie> m_aMovies_PairRDD;  // RDD of movie metrics
private JavaPairRDD<Integer, Customer> m_aCustomers_PairRDD;// RDD of customer metrics
private JavaPairRDD<Integer, MatrixRow> m_aMovieFeatures_PairRDD; // RDD of features by movie
private JavaPairRDD<Integer, MatrixRow> m_aCustFeatures_PairRDD; // RDD of features by customer
private Accumulator<Double> sq_ACC;
private Accumulator<MatrixRow> customerFeatureMatrixRow_ACC, movieFeatureMatrixRow_ACC;
```

```
private Broadcast<MatrixRow> customerFeatureMatrixRow_BRDCST, movieFeatureMatrixRow_BRDCST;
Map<Integer, MatrixRow> mapCustomerFeatures;
Map<Integer, MatrixRow> mapMovieFeatures;
private Broadcast<double[][]> m_aCustFeatures_BRDCST;
private Broadcast<double[][]> m_aMovieFeatures_BRDCST;



public static void Debug(int num) {
   System.out.printf("|----------------------> DBG %d\n", num);
}

public static void Message(String msg) {
    System.out.printf("|-----> %s <----|\n", msg);
}

//---------------------------------------------Constructor-------------
public SVDMovieLensSparkJava() {
int f;
// create Spark Configuration object
conf = new SparkConf().setAppName(this.SparkAppName);
// Register custom classes with Kryo Registrator
//conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer");
//conf.set("spark.kryo.registrator", MyClassRegistrator.class.getName());
// create Spark Context
sc = new JavaSparkContext(this.conf);
TRAINING_FILE_BRDCST = sc.broadcast("/nfs/MovieLens/u.data");
TESTING_FILE_BRDCST = sc.broadcast("/nfs/MovieLens/u1.test");
PREDICTIONS_FILE_BRDCST = sc.broadcast("/nfs/MovieLens/u1.predictions");
MAX_RATINGS_BRDCST = sc.broadcast(100000); // Ratings in entire training set (+1)
MAX_CUSTOMERS_BRDCST = sc.broadcast(943); // Customers in the entire training set (+1)
```

```
MAX_MOVIES_BRDCST = sc.broadcast(1682);    //  Movies  in  the  entire  training  set  (+1)
MAX_FEATURES_BRDCST = sc.broadcast(64);  // Number  of  features  to  use
MIN_EPOCHS_BRDCST = sc.broadcast(120);         // Minimum number of epochs per feature
MAX_EPOCHS_BRDCST = sc.broadcast(200);          // Max  epochs  per  feature
MIN_IMPROVEMENT_BRDCST = sc.broadcast(0.0001);//Minimum improv. req. to contin. cur.f.
INIT_BRDCST  =  sc.broadcast(0.1);    //   Initialization   value   for   features
LRATE_BRDCST  =  sc.broadcast(0.001); // Learning rate  parameter
K_BRDCST = sc.broadcast(0.015); // Regularization parameter used to minimize   over-fitting
// cache broadcast variable values in local variables to speed up int
MAX_FEATURES  =  this.MAX_FEATURES_BRDCST.getValue();
int  MAX_CUSTOMERS  =  this.MAX_CUSTOMERS_BRDCST.getValue();
int  MAX_MOVIES  =  this.MAX_MOVIES_BRDCST.getValue();
Double  INIT  =  this.INIT_BRDCST.getValue();
// Create RDD for Customer Feature matrix, using autogenerated indices
List<MatrixRow> matrixRowList = new ArrayList<>();
for (f = 0; f < MAX_FEATURES;  f++)
{
// create a vector of MAX_CUSTOMERS with INIT_BRDCST value
matrixRowList.add(new MatrixRow(Collections.nCopies(MAX_CUSTOMERS, INIT)));
}
JavaRDD<MatrixRow>   rdd1   =   sc.parallelize(matrixRowList);
this.m_aCustFeatures_PairRDD  =  rdd1.zipWithIndex().mapToPair((Tuple2<MatrixRow,  Long>  tuple)  ->
new  Tuple2<Integer,  MatrixRow>(Integer.valueOf(tuple._2.intValue()),  tuple._1));
// Create RDDs for Movie Feature matrix, using autogenerated indices
matrixRowList = new ArrayList<>();
for (f =  0; f <  MAX_FEATURES;  f++) {
// create a vector of MAX_MOVIES with INIT_BRDCST value
matrixRowList.add(new MatrixRow(Collections.nCopies(MAX_MOVIES, INIT)));
}
this.m_aMovieFeatures_PairRDD = sc.parallelize(matrixRowList).zipWithIndex().
mapToPair((Tuple2<MatrixRow, Long> tuple) ->
```

```
new Tuple2<Integer, MatrixRow>(Integer.valueOf(tuple._2.intValue()), tuple._1));
// create maps from the Customer and Movie Feature RDDs
mapCustomerFeatures = new HashMap<>(m_aCustFeatures_PairRDD.collectAsMap());
mapMovieFeatures  =  new  HashMap<>(m_aMovieFeatures_PairRDD.collectAsMap());
// ------------- Initialize serial version members
int  i;
for (f = 0; f < this.MAX_FEATURES; f++) {
    for (i = 1; i < this.MAX_MOVIES + 1; i++) {
        this.m_aMovieFeatures[f][i]  =  this.INIT;
    }
    for (i = 1; i < this.MAX_CUSTOMERS + 1; i++) {
        this.m_aCustFeatures[f][i] =  this.INIT;
    }
 }
}

//------------------------------Main()-----------------------------------
public static void main(String[] args) throws IOException {
LocalTime t1, t2, t3, t4,    t5;

t1  = LocalTime.now();
SVDMovieLensSparkJava engine = new SVDMovieLensSparkJava();
t2   =  LocalTime.now();
System.out.printf("engine construction duration equals %d s\n",Duration.between(t1,t2).getSeconds());
engine.LoadHistory();
t3  = LocalTime.now();
System.out.printf("load history duration equals %d s\n",Duration.between(t2,t3).getSeconds());
engine.CalcFeatures();
t4  = LocalTime.now();
System.out.printf("calculation feature duration equals %d s\n",Duration.between(t3,t4).getSeconds());
engine.ProcessTest();
```

```
t5  = LocalTime.now();
System.out.printf("processing test duration equals %d s\n",Duration.between(t4,t5).getSeconds());
System.out.println("\nDone\n");
}

// METHODS
      public void LoadHistory() throws IOException {
          this.ProcessFile(this.TRAINING_FILE_BRDCST);
      }

private void ProcessFile(Broadcast<String> filename) throws IOException     {
// create initial RDD of string with training   data
JavaRDD<String>    trainingFile    =  this.sc.textFile(filename.value());
// calculate the number of  ratings
this.m_nRatingCount_BRDCST  =  this.sc.broadcast(trainingFile.count());
// create  RDD  for  Customer statistics
JavaRDD<String[]> columnsTrainingFile = trainingFile.map(line -> line.split("\t"));
JavaPairRDD<Integer, Integer> columnsCustomersA_
= columnsTrainingFile.mapToPair((String[] row) -> new Tuple2(Integer.parseInt(row[0]), Integer.
parseInt(row[2])));
JavaPairRDD<Integer, Integer> columnsCustomersA = columnsCustomersA_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer, Integer> columnsCustomersB_
= columnsTrainingFile.mapToPair((String[] row) -> new Tuple2(Integer.parseInt(row[0]), 1));
JavaPairRDD<Integer, Integer> columnsCustomersB = columnsCustomersB_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer, Tuple2<Integer, Integer>> columnsCustomers = columnsCustomersB.
join(columnsCustomersA);
this.m_aCustomers_PairRDD=columnsCustomers.mapToPair((Tuple2<Integer,Tuple2<Integer,Integer>> tuple)->
new  Tuple2<Integer, Customer>(tuple._1, new  Customer(tuple._2._1, tuple._2._2)));
// create RDD for Movie statistics
JavaPairRDD<Integer, Integer> columnsMoviesA_
=   columnsTrainingFile.mapToPair((String[]   row)   ->   new   Tuple2(Integer.parseInt(row[1]),     Integer.
```

```java
parseInt(row[2])));
JavaPairRDD<Integer, Integer> columnsMoviesA = columnsMoviesA_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer,  Integer>  columnsMoviesB_
= columnsTrainingFile.mapToPair((String[] row) -> new Tuple2(Integer.parseInt(row[1]), 1));
JavaPairRDD<Integer, Integer> columnsMoviesB = columnsMoviesB_.reduceByKey((a, b) -> a + b);
JavaPairRDD<Integer, Tuple2<Integer, Integer>> columnsMovies = columnsMoviesB.
join(columnsMoviesA);
this.m_aMovies_PairRDD = columnsMovies.mapToPair((Tuple2<Integer, Tuple2<Integer, Integer>> tuple) ->
new Tuple2<Integer, Movie>(tuple._1, new Movie(tuple._2._1, tuple._2._2)));
// create Ratings array with Data  objects
JavaRDD<TrainingData> rddData = columnsTrainingFile.map((String[] row) -> new TrainingData(Integer.
parseInt(row[0]), Integer.parseInt(row[1]), Integer.parseInt(row[2])));
JavaPairRDD<TrainingData, Long> rddIndexedData = rddData.zipWithIndex();
this.m_aTrainingRatings_PairRDD = rddIndexedData.mapToPair((Tuple2<TrainingData, Long> tuple) ->
new Tuple2(tuple._2.intValue(), tuple._1));
// ------------------  update  serial  version members
this.m_nRatingCount = this.m_nRatingCount_BRDCST.getValue().intValue();
this.m_aCustomers_PairRDD.collectAsMap().forEach((k, v) -> this.m_aCustomers[k] = v);
this.m_aMovies_PairRDD.collectAsMap().forEach((k, v) -> this.m_aMovies[k] = v);
this.m_aTrainingRatings_PairRDD.collectAsMap().forEach((k, v) -> this.m_aRatings[k] = v);

}



// Calculate features matrices (Cannot be parallelized!) - Driver only
void CalcFeatures() {
    int f, e, i, custId;
    TrainingData rating;
    double err, p, sq, rmse_last = 0.0, rmse = 2.0;
    int movieId;
    double cf, mf;
```

```
for (f = 0; f < this.MAX_FEATURES; f++) {
System.out.printf("--- Calculating feature: %d ---\n", f);
// Keep looping until you have passed a minimum number
// of epochs or have stopped making significant progress
  for (e = 0; (e < this.MIN_EPOCHS) || (rmse <= rmse_last - this.MIN_IMPROVEMENT); e++) {
        sq = 0;
        rmse_last = rmse;
          for (i = 0; i < this.m_nRatingCount; i++) {
              rating = this.m_aRatings[i];
              movieId = rating.MovieId;
              custId = rating.CustId;
              // Predict rating and calc error
              p = PredictRating(movieId, custId, f, rating.Cache, true);
              err = (1.0 * rating.Rating - p);
              sq += err * err;
              // Cache off old feature values
              cf = m_aCustFeatures[f][custId];
              mf = m_aMovieFeatures[f][movieId];
              // Cross-train the features
              m_aCustFeatures[f][custId] += (LRATE * (err * mf - K * cf));
              m_aMovieFeatures[f][movieId] += (LRATE * (err * cf - K * mf));
          }
        rmse = Math.sqrt(sq / m_nRatingCount);
       }
  // Cache off old predictions
  for (i = 0; i < this.m_nRatingCount; i++) {
  rating = m_aRatings[i];
  rating.Cache = PredictRating(rating.MovieId, rating.CustId, f, rating.Cache, false);
  }
}
```

```
        // initialize broadcast matrices to make them available in executors
        m_aCustFeatures_BRDCST = sc.broadcast(m_aCustFeatures);
        m_aMovieFeatures_BRDCST  = sc.broadcast(m_aMovieFeatures);
    }

    // Reads in parallel the test file and produces the predictions file in parallel
    void ProcessTest() throws IOException {
      long cnt; // number of test  ratings
      double  sum;  // sum  of  Abs  difference  between  Rating  and  PredictionRating
      // read test file and create RDD of TrainingData items
      JavaRDD<TrainingData>   c_testingData_RDD
      = this.sc.textFile(this.TESTING_FILE_BRDCST.getValue()).
      map(line    ->    line.split("\t")).
map((String[]  row) ->
new   TrainingData(Integer.parseInt(row[0]),   Integer.parseInt(row[1]),   Integer.parseInt(row[2])));
      // transform RDD of TrainingData to RDD of TestingData items with our predictions
      JavaRDD<TestingData>   testingDataRDD
                =   c_testingData_RDD.
                map((TrainingData  t)  ->  new  TestingData(t,  PredictRating(t.CustId,  t.MovieId)));
        // count test ratings
        cnt   =  testingDataRDD.count();
        // sum the abs differences between ratings and predictionratings
        sum = testingDataRDD.map(t -> t.diff()).reduce((a, b) -> a + b);
        // save RDD of TestingData to prediction file
        testingDataRDD.saveAsTextFile(this.PREDICTIONS_FILE_BRDCST.getValue());
System.out.printf("\n--#621464--\nNumber of predictions:%d\nAvg Abs(diff) : %f\n",cnt,sum/cnt);
    }

    // Used by sequential computations in  CalcFeatures()
   double PredictRating(int movieId, int custId, int feature, double cache, boolean bTrailing)   {
        // Get cached value for old features or default to an   average
```

```
            double sum = (cache > 0) ? cache : 1;
            // Add contribution of current feature
            sum += this.m_aMovieFeatures[feature][movieId] * m_aCustFeatures[feature][custId];
            if (sum > 5) {
                sum = 5;
            }
            if (sum < 1) {
                sum = 1;
            }
            // Add up trailing defaults values
            if (bTrailing) {
                sum += (MAX_FEATURES - feature - 1) * (INIT * INIT);
                if (sum > 5) {
                    sum = 5;
                }
                if (sum < 1) {
                    sum = 1;
                }
            }
            return sum;
        }

        // Used by RDD transformation
    double PredictRating(int custId, int movieId) {
        double sum = 1;
        int f = 0;

        for (f = 0; f < this.MAX_FEATURES_BRDCST.getValue(); f++) {
sum+=this.m_aMovieFeatures_BRDCST.getValue()[f][movieId]*this.m_aCustFeatures_BRDCST.
getValue()[f][custId];
          if (sum > 5) {
```

```
                sum  = 5;
            }
            if (sum  <  1) {
                sum  = 1;
            }
        }
        return sum;
    }
}
```