
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ


# Ανάλυση επίδοσης και μοντελοποίηση του αλγορίθμου συσταδοποίησης k-means σε κεντρικό και κατανεμημένο περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ


**Γρηγόριος Α. Αφεντουλίδης**

**Επιβλέπων :** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.



Αθήνα, Ιούλιος 2015

Η σελίδα αυτή είναι σκόπιμα λευκή

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

# Ανάλυση επίδοσης και μοντελοποίηση του αλγορίθμου συσταδοποίησης k-means σε κεντρικό και κατανεμημένο περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Γρηγόριος Α. Αφεντουλίδης**

**Επιβλέπων :** Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Ιουλίου 2015.

| ..... | ..... | ..... |
|---|---|---|
| Νεκτάριος Κοζύρης<br>Καθηγητής Ε.Μ.Π. | Νικόλαος Παπασπύρου<br>Αν. Καθηγητής Ε.Μ.Π. | Τσουμάκος Δημήτριος<br>Επίκουρος Καθηγητής<br>Ιονίου Πανεπιστημίου |

Αθήνα, Ιούλιος 2015

……...
Γρηγόριος Α. Αφεντουλίδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

# ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια αυξάνεται ραγδαία το ενδιαφέρον για την εκτέλεση data analytics εργασιών τόσο στον επιστημονικό και τεχνολογικό αλλά και στον επιχειρηματικό τομέα. Το φαινόμενο αυτό οδήγησε στην ανάπτυξη μηχανών εκτέλεσης που προσφέρονται ως υπηρεσίες σε παρόχους IaaS και αναλαμβάνουν την διεκπεραίωση τέτοιων εργασιών. Καθώς όμως οι μηχανές αυτές εισάγουν διαφορετικά χαρακτηριστικά και αρχιτεκτονικές εκτέλεσης, υπάρχει η ανάγκη να αναλύσουμε τόσο τους απαιτούμενους υπολογιστικούς πόρους που αυτές χρειάζονται, αλλά και τη χρονική επίδοση που αυτές επιτυγχάνουν. Αυτό μάλιστα γίνεται ακόμα πιο περίπλοκο εφ'όσον επηρεάζεται και από τις παραμέτρους εκτέλεσης των αλγορίθμων που υλοποιούν τις εν λόγω εργασίες. Η αξία της ανάλυσης αυτής, έγκειται στο γεγονός ότι θα προσφέρει τα εφόδια να αναγνωρίσουμε τα πλεονεκτήματα της κάθε μηχανής υπό συγκεκριμένες συνθήκες, ενώ ταυτόχρονα θα ικανοποιήσουμε πολιτικές χρηστών σε περιβάλλοντα cloud που σχετίζονται με το κόστος και την ταχύτητα εκτέλεσης.

Για το σκοπό αυτό είναι αναγκαία η πειραματική ανάλυση των μηχανών εκτέλεσης μέσω μιας διαδικασίας profiling όπου θα μετρούμε τη χρήση των υπολογιστικών πόρων καθώς και τη διάρκεια εκτέλεσης για προσεκτικά επιλεγμένα δείγματα εκτέλεσης. Με τα αποτελέσματα αυτά μπορούμε να κατασκευάσουμε στατικά μοντέλα που να προσομοιώνουν την συμπεριφορά των μηχανών για μεταβολή διαφορετικών παραμέτρων εκτέλεσης.

Στη παρούσα διπλωματική εργασία αναλαμβάνουμε να μελετήσουμε τον αλγόριθμο k-means που χρησιμοποιείται για εργασίες συσταδοποίησης δεδομένων, στο κεντρικό περιβάλλον Weka και στο κατανεμημένο περιβάλλον Apache Spark. Προτείνουμε και υλοποιούμε δύο αρχιτεκτονικές profiling για την ανάκτηση μετρικών που σχετίζονται με τη χρήση των υπολογιστικών πόρων και της χρονική επίδοση κάθε πειραματικής εκτέλεσης. Αναλύουμε από τα αποτελέσματα των μετρήσεών μας τις συμπεριφορές εκτέλεσης των δύο μηχανών καθώς μεταβάλλουμε τις παραμέτρους εκτέλεσης και αναδεικνύουμε πλεονεκτήματα και μειονεκτήματα αυτών. Χρησιμοποιούμε τα δεδομένα που συλλέξαμε για τη κατασκευή μοντέλων για κάθε μετρική και μηχανή εκτέλεσης και αποδεικνύουμε την ακρίβεια καθώς και τη χρησιμότητα αυτών ως προβλεπτικά μοντέλα. Ελέγχουμε την κλιμακωσιμότητα του αλγορίθμου στη κατανεμημένη εκδοχή για διαφορετικό μέγεθος cluster και παρατηρούμε χρονική βελτίωση που αγγίζει το 30%. Τέλος επιχειρούμε σύγκριση των δύο μηχανών εκτέλεσης που μας αναδεικνύει την υπεροχή του Spark ακόμα και για πολύ μικρά μεγέθη dataset.

## Λέξεις κλειδιά

Αλγόριθμος K-means, συσταδοποίηση, Hadoop, Weka, Apache Spark, συστήματα profiling, προβλεπτική μοντελοποίηση

# ABSTRACT

In recent years, interest in execution of data analytics jobs in the science, technology and business fields as well, skyrockets. That led in the development of execution engines that are now offered as services in IaaS providers and take on the execution of such jobs. Meanwhile these execution engines introduce diverse characteristics and runtime architectures, so there exists the need to analyze not only the resources they require but also the execution time they manage to achieve. The complexity of the goal, raises significantly from the fact that this is also affected by the execution parameters of the underlying algorithm. The outcome of such analysis will provide us the means to understand the advantages and disadvantages of every execution engine under specific circumstances and also let us deploy user policies in cloud environments that relate to the cost and the time restraints of the executions.

For this purpose we must conduct an experimental analysis on those engines through a profiling process, where we will measure the usage of the resources as well as the overall execution time with carefully selected execution samples. The results of this process will enable us to construct static predictive models that could simulate the performance of the engines for varying execution parameters.

In this diploma thesis we are studying the k-means algorithm, which is used for clustering jobs, on the centralized environment of Weka and the distributed environment of Apache Spark. We suggest and deploy two profiling architectures that handle the collection of the metrics regarding the resources used and the time of every experimental execurion. We analyze, with the help of the results of the profiling procedure, the performance of those two engines as the execution parameters vary, and we mention the advantages and disadvantages we notice. We also use the collected data for the construction of predictive models for every metric and every engine, and we comment on the accuracy of those models as well as the usefullness they provide. We alter the size of the cluster in the distributed version in order to check the scalability of the algorithm and we notice up to 30% time improvement. Finally we attempt to contrast the two engines, which designates the supremacy of Spark even for very small datasets.



## Keywords

K-means algorithm, clustering, Hadoop, Weka, Apache Spark, profiling systems, predictive modelling

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα καθηγητή, κύριο Νεκτάριο Κοζύρη, για την ευκαιρία που μου έδωσε να ασχοληθώ με το παρόν θέμα στα πλαίσια της διπλωματικής μου εργασίας και για την αμέριστη υποστήριξη και εμπιστοσύνη που μου παρείχε τόσο για την ολοκλήρωση των προπτυχιακών μου σπουδών αλλά και για τη συνέχεια.

Επίσης, θα ήθελα να ευχαριστήσω την μεταδιδακτορική ερευνήτρια του εργαστηρίου Υπολογιστικών Συστημάτων, κυρία Κατερίνα Δόκα που με υπομονή και επιμονή μου παρείχε όλα τα εφόδια για την ολοκλήρωση της παρούσας εργασίας, καθώς και τους υποψήφιους διδάκτορες Γιάννη Γιαννακόπουλο και Χρήστο Μαντά για την πολύ σημαντική τους βοήθεια σε τεχνικά ζητήματα που αντιμετώπισα.

Θέλω ακόμα να ευχαριστήσω του γονείς μου και τις αδερφές μου, η υποστήριξή τους και η αγάπη τους με καθοδήγησε καθόλη τη διάρκεια των σπουδών μου.

Τέλος δεν θα μπορούσα να ευχαριστήσω αρκετά την Άννα, που μαζί μοιραστήκαμε όλα τα νεανικά μας χρόνια. Με τη παρουσία της, οι όμορφες στιγμές ήταν ομορφότερες και οι δύσκολες πιο υποφερτές.

Σας είμαι ευγνώμων.

# Περιεχόμενα

# Κατάλογος Σχημάτων

# Κατάλογος Πινάκων

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Κίνητρο

Το πεδίο της ανάλυσης δεδομένων (data analytics) μας προσφέρει τη δυνατότητα, μέσω αλγοριθμικών εργαλείων και τεχνικών, να επεξεργαστούμε σύνολα δεδομένων και να εξάγουμε μοτίβα και χρήσιμες πληροφορίες από αυτά. Τα αποτελέσματα μιας τέτοιας ανάλυσης έχουν αποδειχτεί ιδιαίτερα επωφελή σε πολλούς τομείς όπως στην ανάλυση ρίσκων, την φαρμακευτική, την οικονομική ανάλυση αγορών και την εξαγωγή γνώσης για την κατασκευή βέλτιστων επιχειρηματικών πλάνων, ανάμεσα σε πολλούς άλλους. Τα πλεονεκτήματα αυτά γίνονται ακόμα πιο πρόδηλα καθώς έχουμε εισέλθει στην εποχή των δεδομένων [1] και παρατηρούμε την ανάπτυξη πολλών και διαφορετικών μηχανών εκτέλεσης που ως στόχο τους έχουν την υλοποίηση τέτοιων data analytics εργασιών με τη χρήση γνωστών αλγορίθμων.

Πολλές από αυτές τις μηχανές προσφέρονται πλέον ως υπηρεσίες από παρόχους IaaS [2][3] δίνοντας ελευθερία επιλογής στον ενδιαφερόμενο. Σε αυτά τα περιβάλλοντα, σημαντικά κριτήρια που κρίνουν την απόφαση επιλογής είναι από τη μία το ποια μηχανή εκτέλεσης κάνει βέλτιστη χρήση των υπολογιστικών πόρων και από την άλλη η ικανοποίηση ενδεχόμενων πολιτικών που ορίζονται από το χρήστη (π.χ βελτιστοποίηση χρονικής επίδοσης). Η πληθώρα των διαφορετικών μηχανών εκτέλεσης σε συνδυασμό με τα διαφορετικά πλάνα εκτέλεσης που αυτές προτείνουν και τις διαφορετικές υλοποιήσεις αλγορίθμων που παρέχουν, δυσχεραίνουν ακόμα περισσότερο αυτή τη διαδικασία επιλογής και αυξάνουν την πολυπλοκότητα.

Για τη διευθέτηση του προβλήματος της βέλτιστης επιλογής για την ικανοποίηση των παραπάνω κριτηρίων και με δεδομένο ότι δεν υπάρχει μία συγκεκριμένη πλατφόρμα που να ικανοποιεί αυτές τις απαιτήσεις[4] , έχουν προταθεί συστήματα [5] που μοντελοποιούν το κόστος και την επίδοση εκτέλεσης data analytics εργασιών και των αλγορίθμων που τις πραγματοποιούν σε διαφορετικές μηχανές εκτέλεσης. Σε αυτά, αναγκαία είναι η ύπαρξη στατικών μοντέλων που δημιουργούνται από τη μελέτη εκτέλεσης αλγορίθμων σε όλες τις μηχανές κάτω από διαφορετικά περιβάλλοντα εκτέλεσης και διαφορετικές παραμέτρους

εκτέλεσης. Η μελέτη αυτή επικεντρώνεται στη χρήση των υπολογιστικών πόρων της κάθε εκτέλεσης καθώς και της χρονικής τους επίδοσης. Για την ανάκτηση αυτών των μετρικών επίσης αναγκαία είναι η κατασκευή συστημάτων profiling που αναλαμβάνουν την πραγματοποίηση δοκιμαστικών εκτελέσεων και την συλλογή των απαιτούμενων μετρικών.

## 1.2 Σκοπός και σύντομη περιγραφή της εργασίας

Στη παρούσα εργασία επιλέγουμε για τη μελέτη μας τον αλγόριθμο συσταδοποίησης k-means και πραγματοποιούμε πειραματικές εκτέλεσεις σε δύο διαφορετικές μηχανές εκτέλεσης. Αυτές οι μηχανές εκτέλεσης είναι το Weka που προσφέρει μια κεντρική υλοποίηση του αλγορίθμου και το Apache Spark που διαθέτει μια κατανεμημένη εκδοχή του αλγορίθμου μέσω της βιβλιοθήκης MLlib.

Οι προαναφερθείσες εκτελέσεις γίνονται για διαφορετικές παραμέτρους που σχετίζονται είτε με τα χαρακτηριστικά του dataset εισόδου (πλήθος instances, dimensions), είτε με τον ίδιο τον αλγόριθμο (επαναλήψεις,cluster), είτε με το περιβάλλον εκτέλεσης (cores ανά μηχάνημα, μνήμη ανά μηχάνημα). Σκοπός αυτών των εκτελέσεων είναι η μέτρηση των υπολογιστικών πόρων που χρειάζεται κάθε μηχανή για την περάτωση του αλγορίθμου, όπως αυτά είναι η χρήση μνήμης σε κάθε μηχάνημα και η χρήση του δίσκου, αλλά και της χρονικής επίδοσης κάθε εκτέλεσης.

Η συλλογή των παραπάνω μετρικών έχει διττή σημασία για τη μελέτη που θέλουμε να κάνουμε στη παρούσα εργασία. Σε πρώτη φάση μας ενδιαφέρει η ανάλυση της εκτέλεσης του αλγορίθμου σε κάθε μηχανή σε σχέση με τις μετρούμενες τιμές και σε σχέση με τη μεταβολή των προαναφερθέντων παραμέτρων. Αυτή η ανάλυση μας ενδιαφέρει ώστε να κατανοήσουμε την πορεία εκτέλεσης και τα πλεονεκτήματα και μειονεκτήματα κάθε μηχανής σε κάθε περίπτωση. Σε δεύτερη φάση οι μετρήσεις αυτές είναι απαραίτητες για την κατασκευή των μοντέλων που θέλουμε για κάθε μηχανή και για κάθε μετρική, τα οποία προσομοιώνουν τη συμπεριφορά εκτέλεσης. Εκτός των άλλων μας ενδιαφέρει η συγκριτική ανάλυση των δύο μηχανών και η πιθανή εύρεση συνθηκών εκτέλεσης που να ευνοούν την χρήση της μίας σε σχέση με την έταιρη.

Για τους παραπάνω σκοπούς προτείνουμε δύο αρχιτεκτονικές συστημάτων profiling, μία κεντρική για το Weka και μία κατανεμημένη για το Spark, με σκοπό την ακριβή μέτρηση και συλλογή των μετρικών που μας ενδιαφέρουν. Στα πλαίσια αυτών των συστημάτων είναι εκτός των άλλων η μορφοποίηση των μετρικών και η τροφοδότησή τους στα κατάλληλα μοντέλα.

Σχετικά με τη διαδικασία εξαγωγής μοντέλων, χρησιμοποιούμε μία πληθώρα διαφορετικών μοντέλων που μας παρέχει το Weka και επικεντρωνόμαστε σε εκείνα με την καλυτερη δυνατή ακρίβεια. Επαληθεύουμε την τελευταία με τον υπολογισμό των μέσων σχετικών σφαλμάτων και των μέσων απόλυτων σφαλμάτων και οπτικοποιούμε την επίδοση του κάθε μοντέλου καθώς μεταβάλλουμε κάθε φορά μόνο μία από τις παραμέτρους εκτέλεσης. Τελικός στόχος είναι η ανάδειξη ενός ή και περισσότερων μοντέλων που να προσεγγίζουν ικανοποιητικά τις μετρήσεις για διαφορετικές εκτελέσεις.

## 1.3 Οργάνωση του κειμένου

Στο Κεφάλαιο 2 παραθέτουμε το γενικό θεωρητικό υπόβαθρο που σχετίζεται με τον αλγόριθμο k-means, τις μηχανές εκτέλεσης που χρησιμοποιούμε στη παρούσα εργασία για την υλοποίηση του αλγορίθμου, καθώς και θεωρητικές γνώσεις για τα μοντέλα που θα χρησιμοποιήσουμε. Τα παραπάνω κρίνονται αναγκαία για τη μελέτη που θα ακολουθήσουμε στις επόμενες ενότητες της εργασίας.

Στο Κεφάλαιο 3 παρουσιάζουμε την κεντρική και κατανεμημένη αρχιτεκτονική profiling που χρησιμοποιήσαμε για την διεξαγωγή των πειραματικών εκτελέσεων που επιχειρήσαμε. Αναλύουμε τη πορεία που ακολουθήσαμε για την συλλογή των μετρήσεών μας και την μορφοποίησή τους, ενώ επίσης παρουσιάζουμε και τα εργαλεία με τα οποία το επιτύχαμε.

Στο Κεφάλαιο 4 επικεντρωνόμαστε στη μελέτη των εκτελέσεών μας στο Weka. Αναφέρουμε αρχικά την πειραματική διαδικασία που ακολουθήθηκε όσον αφορά τις επιλεγμένες εκτελέσεις που δοκιμάσαμε. Παρουσιάζουμε επίσης τα αποτελέσματα των μετρήσεών μας για όλες τις εκτελέσεις και αναλύουμε την συμπεριφορά εκτέλεσης καθώς μεταβάλλουμε κάθε παραμέτρο εκτέλεσης. Τέλος κατασκευάζουμε τα μοντέλα για κάθε μετρική και παρουσιάζουμε τα αποτελέσματα της μοντελοποίησης.

Στο Κεφάλαιο 5 ακολουθούμε την ίδια ακριβώς πορεία με αυτή του Κεφαλαίου 4 για την κατανεμημένη υλοποίηση του αλγορίθμου στο Apache Spark. Παραθέτουμε τις διαφορετικές εκτελέσεις που συνθέτουν τη πειραματική διαδικασία, παρουσιάζουμε τα αποτελέσματα των μετρήσεών μας για κάθε μετρική και τέλος παρουσιάζουμε τη διαδικασία κατασκευής μοντέλων για την μηχανή Spark. Επίσης αφιερώνουμε μία ενότητα που σχολιάζουμε συγκριτικά αποτελέσματα μεταξύ των δύο υλοποιήσεων και μία ακόμα που αφορά την κλιμακωσιμότητα του αλγορίθμου για διαφορετικά μεγέθη cluster.

Στο Κεφάλαιο 6 ανακεφαλαιώνουμε τα συμπεράσματα που εξάγαμε από όλη την μελέτη μας και σχολιάζουμε τυχόν επεκτάσεις της παρούσας εργασίας που θα μπορούσαν να γίνουν μελλοντικά.

# Κεφάλαιο 2

# Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό, κάνουμε μία μικρή εισαγωγή στην έννοια της συσταδοποίησης ως μιας διαδικασίας με ευρύτατο φάσμα εφαρμογών σε πολλά διαφορετικά επιστημονικά πεδία και παρουσιάζουμε τον αλγόριθμο k-means που την υλοποιεί. Στη συνέχεια κάνουμε μία μικρή επισκόπηση των κατανεμημένων συστημάτων Hadoop και Spark, του κατανεμημένου συστήματος αρχείων HDFS και της βιβλιοθήκης MLlib που προσφέρει υλοποιήση του αλγορίθμου k-means για το Apache Spark. Επίσης παρουσιάζουμε την σουίτα προγραμμάτων Weka που προσφέρει μια κεντρική μη κατανεμημένη υλοποίηση του αλγορίθμου. Στη τελευταία ενότητα του κεφαλαίου κάνουμε μία συνοπτική θεωρητική εισαγωγή στη προβλεπτική μοντελοποίηση και αναφερόμαστε επίσης στα μοντέλα που χρησιμοποιήσαμε στην παρούσα εργασία.

## 2.1 Συσταδοποίηση

### 2.1.1 Γενικά

Η έννοια της συσταδοποίησης προκύπτει φυσικά στα προβλήματα εκείνα κατά τα οποία δεδομένου ενός συνόλου αντικειμένων μας ενδιαφέρει να δημιουργήσουμε επιμέρους ομάδες μέσα στις οποίες τοποθετούμε αντικείμενα που ικανοποιούν μια έννοια ομοιότητας. Αποτελεί μία από τις πιο συνηθισμένες μορφές μη επιβλεπόμενης μάθησης, όπου απουσία επίβλεψης εννοείται η μη πρότερη γνώση ύπαρξης σαφώς ορισμένων ομάδων μεταξύ των υπό εξέταση αντικειμένων. Συχνά η συσταδοποίηση είναι μία προκαταρτική διαδικασία που μας προσφέρει χρήσιμη γνώση για επόμενα στάδια ανάλυσης σε χώρους αντικειμένων με μικρή πρότερη εμπειρία.

Παρότι η κατηγοριοποίηση των αντικειμένων του φυσικού κόσμου από τους ανθρώπους είναι μια υποσυνείδητη και αυτοματοποιημένη διαδικασία, στον κόσμο των υπολογιστών και της μηχανικής μάθησης οι έννοιες αντικείμενα και ομοιότητα χρειάζονται μαθηματική μοντελοποίηση. Αυτή αφορά συνήθως στην αναπαράσταση των αντικειμένων που μας ενδιαφέρουν σε διανύσματα στο ν-διάστατο ευκλείδιο χώρο όπου η πληθικότητα των διαστάσεων του διανύσματος αντικατοπτρίζει τον αριθμό των χαρακτηριστικών-

ιδιοτήτων που περιγράφουν το εκάστοτε αντικείμενο, ενώ οι αντίστοιχες τιμές αυτών τα ποσοτικοποιούν σε μια επιλεγμένη κλίμακα. Καθώς λοιπόν τα αντικείμενα αποτελούν διανύσματα στο χώρο η έννοια της ομοιότητας αντιστοιχίζεται εύλογα με κάποια νόρμα του ευκλείδιου χώρου. Ανάμεσα στις πολλές επιλογές οι συνηθέστερες αφορούν την ευκλείδια απόσταση, την απόσταση manhattan και την απόσταση συνημιτόνου. Οι εξισώσεις υπολογισμού των παραπάνω φαίνονται παρακάτω για δεδομένα διανύσματα X και Y.

$$d_{Euclidean}(X,Y) = \| X - Y \|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots}$$

$$d_{Manhattan}(X,Y) = \| X - Y \|_1 = | x_1 - y_1 | + | x_2 - y_2 | + \ldots$$

$$d_{cosine}(X,Y) = 1 - \frac{X \cdot Y}{\| X \|_2 - \| Y \|_2}$$

Ανεξάρτητα από την μοντελοποίηση και την επιλογή συνάρτησης απόστασης, η ίδια η συσταδοποίηση κατηγοριοποιείται σε επίπεδη και ιεραρχική ανάλογα με το τελικό αποτέλεσμα. Συγκεκριμένα η επίπεδη συσταδοποίηση αναφέρεται σε πλήρως ανεξάρτητες συστάδες αντικειμένων με μηδενική αλληλεξάρτηση που στόχο έχει μόνο την ανάδειξη ομοιοτήτων αντικειμένων-μελών μόνο της ίδιας συστάδας. Η δεύτερη κατηγορία συσταδοποίησης στοχεύει στην εμφώλευση συστάδων σε δενδρική μορφή, όπου στη ρίζα έχουμε συστάδες που περιγράφουν γενικευμένες ομαδοποιήσεις ενώ οι ενδιάμεσοι κόμβοι αποτελούν πιο εξειδικευμένες ομαδοποιήσεις. Η τελευταία κατηγορία οπτικοποιείται ιδανικά με τα λεγόμενα δενδρογράμματα.

Επιπρόσθετα οι συσταδοποιήσεις μπορούν να χαρακτηριστούν ολικές ή μερικές. Στις ολικές συσταδοποιήσεις ο τελικός σκοπός είναι κάθε ένα από τα αντικείμενα εισόδου της διαδικασίας να αποτελούν μέλος μιας συστάδας. Αντίθετα στις μερικές μπορεί να υπάρξουν και αντικείμενα τα οποία δεν εμπεριέχονται σε κάποια συστάδα μετά το πέρας της συσταδοποίησης. Η τελευταία κατηγορία αποτελεί λογική επιλογή όταν θέλουμε να απομονώσουμε αντικείμενα του χώρου που είναι αισθητά διαχωρίσιμα από τα υπόλοιπα και λειτουργούν σαν ένα είδος θορύβου.

Ακόμα ένα στοιχείο που κατηγοριοποιεί την διαδικασία, είναι ο τρόπος που αντιλαμβανόμαστε την έννοια μιας συστάδας. Προς αυτή τη κατεύθυνση έχουμε συστάδες όπου εκπροσωπούνται από αντιπροσώπους (prototype-based), οι οποίες αποτελούνται από αντικείμενα που βρίσκονται πιο κοντά στο συγκεκριμένο αντιπρόσωπο σε σχέση με αυτούς των άλλων συστάδων. Επίσης έχουμε συστάδες που δημιουργούνται από αντικείμενα που βρίσκονται πιο κοντά σε τουλάχιστον ένα αντικείμενο της ίδιας συστάδας σε σχέση με οποιοδήποτε άλλο σε έταιρη συστάδα (continuity-based). Επιπρόσθετα μπορεί να έχουμε συστάδες που βασίζονται στη πυκνότητα αντικειμένων στο χώρο που εξετάζουμε (density-based), σε αυτή τη τελευταία περίπτωση, συστάδα αντικειμένων θεωρούμε μια περιοχή με υψηλή περιεκτικότητα και εκτός αυτής αισθητά χαμηλότερη περιεκτικότητα.

Από τα παραπάνω φαίνεται πως η συσταδοποίηση έχει στενή εξάρτηση με το είδος και τα ιδιαίτερα χαρακτηριστικά του συνόλου των αντικειμένων στο οποίο θα εφαρμοστεί. Απόρροια αυτών είναι και η ύπαρξη μιας πληθώρας αλγορίθμων, με πολύ διαφορετικές πολυπλοκότητες, που την υλοποιούν. Η γενικότερη αξία και αναγκαιότητα της διαδικασίας πάντως είναι και αυτή που την καθιστά χρήσιμη σε ένα μεγάλο εύρος επιστημονικών πεδίων όπως η μηχανική εκμάθηση, η αναγνώριση προτύπων, η ανάλυση εικόνας και βίντεο, η εξόρυξη δεδομένων, η βιοπληροφορική και πολλά άλλα.

Στην παρούσα εργασία θα εστιάσουμε το ενδιαφέρον μας σε έναν πολύ γνωστό αλγόριθμο συσταδοποίησης που ονομάζεται κ-μέσοι (k-means), τον οποίο παρουσιάζουμε στην αμέσως επόμενη υποενότητα.

### 2.1.2 Αλγόριθμος k-means

Ο αλγόριθμος k-means προτάθηκε πρώτη φορά από τον Stuart Lloyd το 1957 ως μια τεχνική για διαμόρφωση παλμικού κώδικα (pulse code modulation) για αυτό και πολλές φορές συναντάται και με το όνομα αλγόριθμος του Lloyd. Ο αλγόριθμος εκτελεί συσταδοποίηση αντικειμένων στον ν-διάστατο ευκλείδιο χώρο ενώ συνηθέστερα η ευκλείδια νόρμα χρησιμοποιείται ως συνάρτηση απόστασης.

Ανήκει στη κατηγορία των αλγορίθμων που εκτελούν επίπεδη, ολική συσταδοποίηση ενώ η αναπαράσταση των συστάδων ανήκει στην κατηγορία prototype-based που αναφέραμε στην προηγούμενη ενότητα. Συγκεκριμένα ο αντιπρόσωπος κάθε συστάδας θεωρείται το λεγόμενο κεντροειδές (centroid) που υπολογίζεται ως το διάνυσμα εκείνο που είναι η μέση τιμή των διανυσμάτων που αποτελούν τη συγκεκριμένη συστάδα.

Ο αλγόριθμος, δεδομένου ενός συνόλου n αντικειμένων κατάλληλα μοντελοποιημένων ως διανυσμάτων με διαστάσεις d που αναπαριστούν τα χαρακτηριστικά τους, εκτελεί επαναληπτικά τις παρακάτω 2 φάσεις:

- Αντιστοίχηση αντικειμένων σε συστάδα
- Επαναϋπολογισμός κεντροειδών

Για την φάση της αντιστοίχισης αντικειμένων σε συστάδα κατά την εκκίνηση του αλγορίθμου επιλέγουμε κάποια αντικείμενα του χώρου, συνήθως τυχαία, τα οποία αναλαμβάνουν τον ρόλο των αρχικών κεντροειδών. Σε δεύτερο χρόνο ο αλγόριθμος αποφασίζει για κάθε αντικείμενο εισόδου την απόστασή του με κάθε ένα από τα κεντροειδή σύμφωνα με την ευκλίδεια απόσταση, και αναλαμβάνει να αντιστοιχίσει το κάθε ένα από αυτά στο κοντινότερο κεντροειδές. Στο τέλος της πρώτης φάσης κάθε επανάληψης το αποτέλεσμα είναι να έχουμε k συστάδες που αντιπροσωπεύονται από τα k κεντροειδή και αποτελούνται από αντικείμενα τα οποία βρίσκονται εγγύτερα στο κεντροειδές-εκπρόσωπο της εκάστοτε συστάδας σε σχέση με κάθε άλλο κεντροειδές.

Η φάση του επαναϋπολογισμού κεντροειδών αφορά στην εύρεση του αριθμητικού μέσου των αντικειμένων-διανυσμάτων που αποτελούν την κάθε συστάδα έτσι όπως αυτές διαμορφώθηκαν από το προηγούμενο βήμα του αλγορίθμου και στην ανακύρηξη αυτού του αποτελέσματος ως νέου κεντροειδούς. Με το πέρας και αυτής της φάσης έχουμε k νέα κεντροειδή τα οποία επανατροφοδοτούνται στην πρώτη φάση της επόμενης επανάληψης.

Ως συνθήκη τερματισμού του αλγορίθμου υπάρχουν πολλές διαφορετικές επιλογές. Ίσως η πιο απλή αφορά στον αριθμό των επαναλήψεων που θα εκτελέσει ο αλγόριθμος, σύμφωνα με την οποία μπορούμε να διαλέξουμε έναν αριθμό επαναλήψεων που πιστεύουμε ότι θα παράξει μία ικανοποιητική συσταδοποίηση και εκεί να τερματίσουμε τον αλγόριθμο. Ακόμα μία επιλογή είναι να θεωρήσουμε ένα συγκεκριμένο κατώφλι ε που να εκφράζει την ελάχιστη απόσταση που αναμένουμε να έχουν μετακινηθεί τα κεντροειδή σε κάθε επανάληψη. Σε περίπτωση που ο αλγόριθμος βρεθεί στη συνθήκη κατά την οποία κανένα κεντροειδές δεν μετακινήθηκε περισσότερο από το ζητούμενο κατώφλι ο αλγόριθμος τερματίζει. Ένας ακόμα τρόπος να εκφράσουμε την ποιότητα μιας συστάδας είναι το άθροισμα των τετραγώνων των αποστάσεων όλων των διανυσμάτων μιας συστάδας με το κεντροειδές που την αντιπροσωπεύει (RSS – Residual Sum of Squares). Δεδομένου ότι κάθε διαμορφωμένη συστάδα χαρακτηρίζεται από μία μετρική RSS θα μπορούσαμε και πάλι να ορίσουμε ένα ανώτατο όριο, ώστε αν επιτυγχάναμε RSS κάτω από το αυτό για κάθε συστάδα να τερματίζαμε τον αλγόριθμο.

Όσο αναφορά στη χρονική πολυπλοκότητα του αλγορίθμου, αυτή υπολογίζεται εύκολα και είναι $O(nkdi)$, όπου θεωρούμε το n ως το πλήθος των αντικειμένων-διανυσμάτων που συμμετέχουν στη διαδικασία, k των αριθμό των συστάδων που ζητάμε, d των πλήθος των διαστάσεων του κάθε διανύσματος και τέλος με i των αριθμό των επαναλήψεων. Στις συνηθέστερες εφαρμογές τα k,d και i είναι σημαντικά μικρότερα του n οπότε μπορούμε να πούμε ότι ο αλγόριθμος είναι σχεδόν γραμμικός όσο αναφορά το πλήθος των διανυσμάτων εισόδου. Η χωρική πολυπλοκότητα είναι επίσης εύκολα υπολογίσιμη και δίνεται $O((n+k)\cdot d)$, που εκφράζει ακριβώς την ανάγκη του αλγορίθμου να έχει διαθέσιμα στην μνήμη τουλάχιστον όλα τα δεδομένα συν τα κεντροειδή κάθε επανάληψης. Είναι σημαντικό εδώ να αναφέρουμε ότι διάφορες μέθοδοι για την αρχική επιλογή των κεντροειδών έχουν προταθεί [6] καθώς έχει αποδειχθεί ότι η προνοητική και προσεκτική επιλογή τους έχει βαρύνουσα σημασία στο τελικό αποτέλεσμα αλλά και στον χρόνο που επιτυγχάνεται η επιθυμητή σύγκλιση.

Ο αλγόριθμος k-means όπως βλέπουμε αποτελεί μια εύκολη και αξιόπιστη λύση για το πρόβλημα της συσταδοποίησης, και η ελκυστικότητά του αυτή είναι που έχει οδηγήσει σε πολλές παραλαγές του [7] και στην ευρεία χρήση του σε πολλούς διαφορετικούς τομείς. Ακόμα όμως ένα σημαντικό πλεονέκτημα του αλγορίθμου που αξίζει να αναφέρουμε είναι ότι είναι εγγενώς παραλληλοποιήσιμος. Συγκεκριμένα όπως φαίνεται από την πρότερη περιγραφή του αλγορίθμου η διαδικασία υπολογισμού επικεντρώνεται κυρίως στην αντιστοίχηση του κάθε αντικειμένου με το πλησιέστερο κάθε φορά κεντροειδές και είναι ανεξάρτητη με τους αντίστοιχους υπολογισμούς για κάθε άλλο σημείο. Αυτή του η ιδιότητα τον κάνει ιδιαίτερα προσφιλή σε κατανεμημένα συστήματα όπου το σύνολο των δεδομένων εισόδου διαχωρίζεται σε μικρότερα κομμάτια και οι υπολογισμοί αναλαμβάνονται από διαφορετικούς υπολογιστικούς κόμβους. Ο μόνος περιορισμός είναι η απλή ενημέρωση των κόμβων κάθε φορά με το νέο σύνολο κεντροειδών κάτι όμως που πραγματοποείται εύκολα και αποδοτικά από τα σύγχρονα κατανεμημένα προγραμματιστικά μοντέλα. Αυτός είναι και ο λόγος που σε αυτά τα περιβάλλοντα ο αλγόριθμος k-means διαπρέπει σε σχέση με ανταγωνιστικούς αλγορίθμους.

## 2.2 Μηχανές και εργαλεία εκτέλεσης

### 2.2.1 MapReduce

Το MapReduce παρουσιάστηκε από την Google το 2004 [8], και αποτελεί ένα προγραμματιστικό πλαίσιο (framework) που είχε υλοποιήσει και παρουσίασε η εταιρία, για την παράλληλη επεξεργασία και ανάλυση μεγάλων datasets σε συστάδες υπολογιστών (clusters). Το framework εκθέτει μια απλή προγραμματιστική διεπαφή για την συγγραφή κατανεμημένων εφαρμογών που θα εκτελεστούν σε cluster, ενώ παράλληλα παρέχει υπηρεσίες που αναλαμβάνουν την παραλληλοποίηση των υπολογισμών, την επίβλεψη της σωστής εκτέλεσής τους καθώς και την διασφάλιση της ανοχής σε σφάλματα κατά τη διάρκεια του κύκλου εκτέλεσης της εφαρμογής.

Η φιλοσοφία του προγραμματιστικό μοντέλου το οποίο χρησιμοποιεί το MapReduce, χωρίζεται σε δύο ξεχωριστές φάσεις που ονομάζονται map και reduce, από όπου προκύπτει και το όνομα. Κάθε φάση τροφοδοτείται από ένα ζεύγος κλειδιού-τιμής (key-value pair) και αντίστοιχα παρέχει ως έξοδο εκ νέου κάποιο key-value pair. Οι φάσεις map και reduce ορίζονται πλήρως από τον κώδικα του προγραμματιστή μέσω συγγραφής των συνονόματων μεθόδων:

- **Μέθοδος Map:** Η μέθοδος map της εφαρμογής περιγράφει το είδος της επεξεργασίας που επιτελείται σε κάθε εισερχόμενο key-value ζεύγος που έχει δημιουργηθεί από το dataset ειδόδου και αναλαμβάνει την δημιουργία ενός ή περισσότερων νέων ζευγών που υποχρεούται να επιστρέψει.

- **Μέθοδος Reduce:** Η μέθοδος reduce δέχεται ως είσοδο ένα ζεύγος που έχει ως key ένα από τα keys που δημιουργήθηκαν από τη φάση map και ως value μια λίστα από τα values του συγκεκριμένου κλειδιού. Μετά από επεξεργασία η μέθοδος επιστρέφει πάλι ένα η περισσότερα key-value ζεύγη τα οποία αποτελούν και την τελική έξοδο της εφαρμογής.

Η αρχιτεκτονική εκτέλεσης μιας εφαρμογής MapReduce στηρίζεται στο αρχιτεκτονικό μοντέλο master/slave και ακολουθεί τα παρακάτω βήματα, που περιγράφουμε εδώ συνοπτικά:

1. Ο προγραμματιστής γράφει μία μέθοδο map και μία μέθοδο reduce για την εφαρμογή, ενώ επίσης ορίζει και πως από το αρχικό dataset δημιουργούνται τα αρχικά ζεύγη key-value.
2. Ο master επιμερίζει την εφαρμογή σε map και reduce εργασίες (tasks) που τις αναθέτει σε κάθε διαθέσιμο κόμβο του cluster.
3. Τα map tasks εκτελούνται σε κομμάτια του ολικού dataset (splits) και παράγουν τα ενδιάμεσα ζεύγη σύμφωνα με την μέθοδο map. Τα αποτελέσματα αποθηκεύονται στον δίσκο του εκάστοτε κόμβου μέχρι να ολοκληρωθούν όλα τα map tasks.
4. Τα reduce tasks παίρνουν ομαδοποιημένα τα values που αντιστοιχούν σε κοινό key που έχει παραχθεί από τους mapper. Αυτή την ομαδοποίηση την αναλαμβάνει ο partitioner με μία οριζόμενη συνάρτηση διαχωρισμού (partition function).

5. Οι reducers επεξεργάζονται πλέον τα νέα ζεύγη και παράγουν τα τελικά τα οποία είναι και το αποτέλεσμα της εφαρμογής.

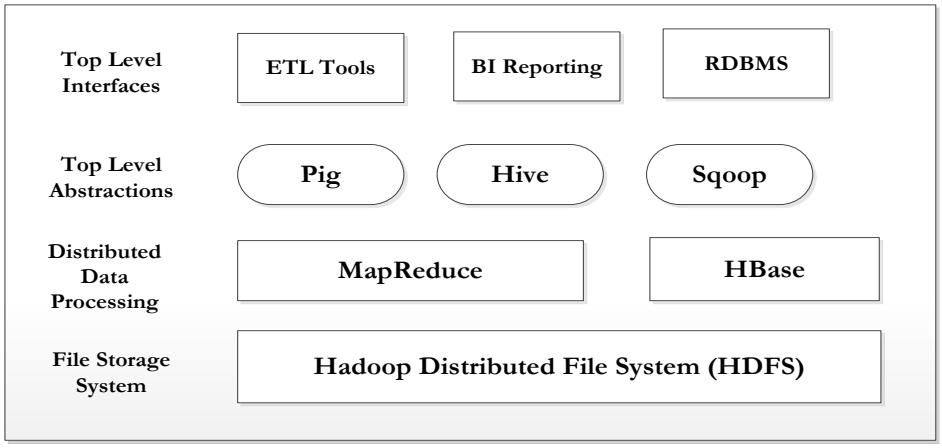
Πέρα από το παραπάνω μοντέλο εκτέλεσης το MapReduce παρέχει, και την ιδιαίτερα επιθυμητή σε κατανεμημένα συστήματα, ανοχή σε σφάλματα τόσο στο επίπεδο της εφαρμογής (σφάλματα σε υπολογισμούς map ή reduce) όσο και στο επίπεδο του υλικού (προβλήματα δικτύου,δίσκου κτλ). Συγκεκριμένα ενσωματώνει όπως αναφέραμε ήδη υπηρεσίες που του επιτρέπουν να ολοκληρώνει τους υπολογισμούς ακόμα και όταν εκτελείται σε αναξιόπιστο περιβάλλον, είτε επαναδρομολογώντας map ή reduce tasks είτε παρακάμπτωντας εντελώς έναν μη διαθέσιμο υπολογιστικό κόμβο του cluster.

Η ιδιαίτερη αξία του MapReduce προκύπτει από το γεγονός ότι προσφέρει ένα απλοϊκό αλλά ταυτόχρονα πολύ ισχυρό μοντέλου προγραμματισμού για κατανεμημένες εφαρμογές, αρκεί αυτές να εκφραστούν στις φάσεις map και reduce. Η συμβολή του στο τομέα των κατανεμημένων συστημάτων αποτυπώνεται στο γεγονός ότι σχεδόν άμεσα υλοποιήθηκε ανοικτού κώδικα ανταγωνιστική υλοποίηση που περιλαμβάνεται στο οικοσύστημα Hadoop, ενώ το μοντέλο εκτέλεσης παραμένει αμετάβλητο και στα νέα συστήματα παράλληλης επεξεργασίας όπως το Apache Spark που θα μελετήσουμε παρακάτω.

## 2.2.2 Hadoop

### 2.2.2.1 Εισαγωγή

Το Hadoop αποτελεί ένα προγραμματιστικό πλαίσιο (software framework) ανοικτού κώδικα το οποίο επιτρέπει των προγραμματισμό και την εκτέλεση κατανεμημένων εργασιών που διαχειρίζονται πολύ μεγάλο όγκο δεδομένων σε συστάδες (clusters) υπολογιστών, χρησιμοποιώντας απλές προγραμματιστικές δομές και μοντέλα. Ο σχεδιασμός του ευνοεί την επεκτασιμότητα από έναν υπολογιστικό κόμβο μέχρι αρκετές χιλιάδες. Ένα ακόμα σημαντικό χαρακτηριστικό του είναι το γεγονός ότι το Hadoop αναλαμβάνει τον εντοπισμό και την διαχείριση των σφαλμάτων που σχετίζονται με το υλικό πάνω στο οποίο εγκαθίσταται (π.χ. προβλήματα δικτύου, αστοχία δίσκου). Από το τελευταίο προκύπτουν και τα δύο πολύ σημαντικά πλεονεκτήματα του Hadoop. Το πρώτο αφορά στη χρησιμοποίηση του framework σε υλικό ανέξαρτητο συγκεκριμένου τύπου προδιαγραφών και άρα κόστους, και το δεύτερο αφορά την αυτοματοποιημένη αντιμετώπιση των αντικειμενικών δυσκολιών που αντιμετωπίζουν οι προγραμματιστές και οι διαχειριστές κατανεμημένων συστημάτων.




*Σχήμα 2.1: Hadoop Ecosystem*

Το Hadoop έχει τις ρίζες του στο Apache Nutch, που ήταν μια ανοικτού κώδικα μηχανή αναζήτησης διαδικτύου που ξεκίνησε να δημιουργεί ο Doug Cutting το 2002. Το εγχείρημα γρήγορα παρουσίασε δυσκολίες που είχαν να κάνουν με τον μεγάλο όγκο δεδομένων που έπρεπε να διαχειρίζεται η μηχανή και τον τρόπο επεξεργασίας τους. Την σημαντικότερη ίσως συμβολή για την σημερινή μορφή του Hadoop είχαν οι δημοσιεύσεις της Google για το Google File System (GFS) [9] το 2003 που αφορούσε την υλοποίηση ενός κατανεμημένου συστήματος αρχείων καθώς και αυτή για το MapReduce που αναφέραμε στη προηγούμενη ενότητα. Το Hadoop ενσωμάτωσε αυτές τις ιδέες και αναπτύχθηκαν με αυτό τον τρόπο το Hadoop Distributed File System (HDFS) καθώς και η ανοιχτού κώδικα εκδοχή του MapReduce, ενώ από το 2008 θεωρείται πλεον top-level project της Apache Software Foundation που είναι και υπεύθυνη για την περαιτέρω εξέλιξή του.

Παρότι το HDFS και το MapReduce έχουν συγκεντρώσει την μεγαλύτερη προσοχή, θα πρέπει να επισημάνουμε ότι αποτελούν δύο από τα υποσυστήματα του οικοσυστήματος, που περιγράφουμε ως Hadoop. Ενδεικτικά εδώ αναφέρουμε ότι υποσυστήματα όπως η Hbase, το Hive, το ZooKeeper, αποτελούν οντότητες που παρέχουν διαφορετικές υπηρεσίες σε κατανεμημένα περιβάλλοντα και συνεργάζονται ιδανικά με το HDFS και το μοντέλο MapReduce που προαναφέραμε. Αυτό ακριβώς το γεγονός αναδεικνύεται και στο Σχήμα 2.1.

## 2.2.2.2 Hadoop Distributed File System (HDFS)

Το Hadoop Distributed File System είναι όπως περιγράφει και το όνομά του η υλοποίηση ενός κατανεμημένου συστήματος αρχείων του Hadoop. Εγκαθίσταται σε cluster υπολογιστών και παρέχει τη δυνατότητα αποθήκευσης πολύ μεγάλων αρχείων δεδομένων που διαμοιράζονται σε όλους τους κόμβους που συμμετέχουν στο hdfs cluster ενώ παράλληλα προσφέρει διεπαφές προγραμματισμού (API) που δίνουν την εντύπωση ενός κλασικού ενιαίου συστήματος αρχείων. Παρακάτω παραθέτουμε συνοπτικά τα βασικότερα χαρακτηριστικά του hdfs:

- **Ανοχή σε Αστοχίες Υλικού:** Οι αστοχίες υλικού σε μεγάλα cluster υπολογιστών είναι ένα από τα κυριότερα προβλήματα με τα οποία έρχονται αντιμέτωποι οι σχεδιαστές

κατανεμημένων εφαρμογών. Στην περίπτωση ενός κατανεμημένου συστήματος αρχείων αυτό μεταφράζεται σε απώλεια δεδομένων. Το hdfs αντιμετωπίζει τέτοια προβλήματα αυτοματοποιημένα εντοπίζοντας αρχικά τον κόμβο όπου έχει δημιουργηθεί το πρόβλημα, ενώ παράλληλα διευθετεί το πρόβλημα της απώλειας δεδομένων κρατώντας εφεδρικά αντίγραφα των δεδομένων που βρίσκονται σε κάθε κόμβο σε έταιρους κόμβους του cluster.

- **Πρόσβαση δεδομένων συνεχούς ροής (streaming data access):** Το hdfs είναι σχεδιασμένο έτσι ώστε να ευνοεί την χρήση του από εφαρμογές που απαιτούν συνεχή ροή δεδομένων μεγάλου μεγέθους. Τέτοιες εφαρμογές απαιτούν μεγάλη διεκπαιρεωτική ικανότητα (throughput) από το σύστημα αρχείων που χρησιμοποιούν.

- **Μεγάλα αρχεία δεδομένων:** Η υλοποίηση του hdfs είναι τέτοια ώστε να αναμένει αρχεία που εκτείνονται σε μέγεθος από αρκετά gigabytes μέχρι terabytes. Η σχεδίαση του hdfs του επιτρέπει να εξυπηρετεί την αποθήκευση πολλών τέτοιων αρχείων αλλά και την αποδοτική διαχείρισή τους.

- **Απλό μοντέλο συνεκτικότητας (Simple coherency model):** Συγκεκριμένα το μοντέλο που υιοθετείται είναι αυτό της μονής εγγραφής-πολλών αναγνώσεων. Δηλαδή το hdfs αναμένει συνήθως μηδαμινή ή μηδενική τροποποίηση των αποθηκευμένων αρχείων αλλά αντίθετα μεγάλο αριθμό αναγνώσεων. Επίσης σε αντίθεση με τα κλασικά συστήματα αρχείων οι εγγραφές περιορίζονται στην εναπόθεση νέων δεδομένων μόνο στο τέλος των υπάρχοντων αρχείων. Αυτό απλοποιεί το μοντέλο συνοχής του συστήματος αρχείων και δίνει προτεραιότητα όπως ήδη αναφέραμε στην ικανότητά του να παρέχει καλό throughput.

- **Ετερογενές λογισμικό και υλικό:** Το hdfs έχει υλοποιηθεί εξ'ολοκλήρου στη γλώσσα προγραμματισμού java. Αυτό του επιτρέπει την ευελιξία να λειτουργεί ανεξαρτήτως του υλικού και του λογισμικού (π.χ λειτουργικό σύστημα). Αυτό δίνει και την ελευθερία για κατασκευή cluster υπολογιστών με πολύ διαφορετικά χαρακτηριστικά ανάμεσα στους εμπλεκόμενους υπολογιστικούς κόμβους και άρα μεγάλη ελευθερία επιλογής τελικού κόστους κατασκευής.

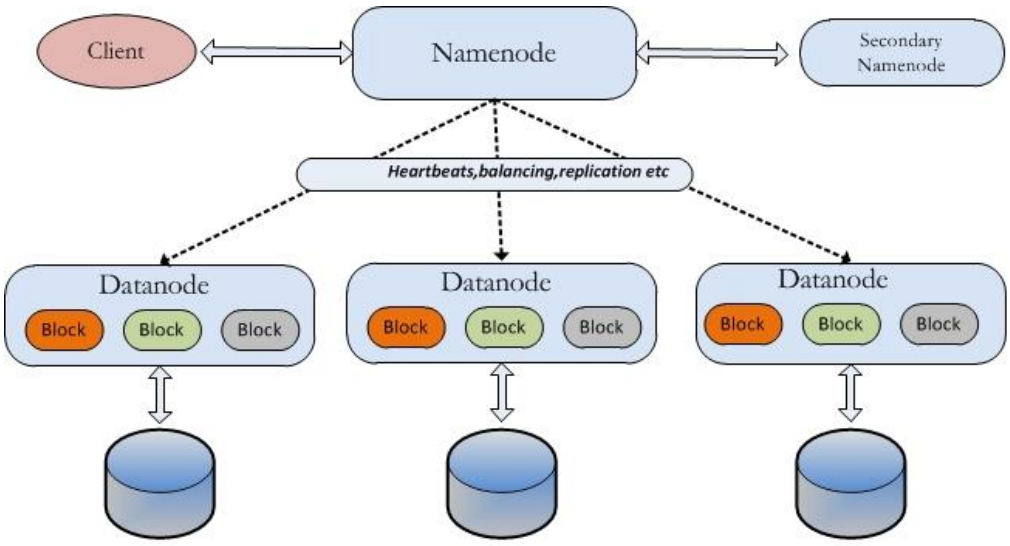
- **Πρόσβαση δεδομένων με χαμηλό latency:** Η ικανότητα υψηλού throughput που θέλει να επιτύχει το σύστημα έρχεται σε σύγκρουση με την βελτιστοποίηση του latency που αποτελεί την λανθάνουσα περίοδο σε μία διαδικασία εγγραφής ή ανάγνωσης στο σύστημα αρχείων. Το hdfs θυσιάζει τον παράγοντα του latency με γνώμονα να εξυπηρετεί εφαρμογές με αντίστοιχες προδιαγραφές.

Η μικρότερη μονάδα αποθήκευσης δεδομένων στο hdfs είναι το block. Ένα αρχείο στο hdfs αποτελείται από ένα σύνολο blocks τα οποία αποθηκεύονται στους κόμβους που αποτελούν το cluster με τρόπο που θα περιγράψουμε σύντομα. Παρότι η ονομασία block είναι αυτή που χρησιμοποιείται για να εκφράσει την ίδια έννοια και στα κλασικά συστήματα αρχείων αλλά και στους δίσκους, στο hdfs υπάρχουν δύο διαφοροποιήσεις με αυτές τις περιπτώσεις. Η πιο σημαντική έχει να κάνει με την επιλογή του μεγέθους του block, η οποία έχει default τιμή 64MB (παρέχεται η δυνατότητα αλλαγής αυτής της τιμής) και ξεπερνάει κατά πολύ τις αντίστοιχες των κεντρικών συστημάτων αρχείων. Αυτή η επιλογή είναι απολύτως λογική δεδομένου ότι το συνολικό μέγεθος ενός αρχείου αναμένουμε να είναι

τάξης μεγέθους gigabyte ή terabyte όπως έχουμε ήδη αναφέρει. Μία δεύτερη διαφοροποίηση είναι ότι όταν ένα αρχείο είναι μικρότερο του διαθέσιμου χώρου που του παρέχει το hdfs block τότε ο επιπλέον χώρος δεν δεσμεύεται από το σύστημα αρχείων.

Στο επίπεδο του block λειτουργούν και δύο ακόμα πολύ σημαντικά χαρακτηριστικά του συστήματος. Αυτά είναι η ανοχή στα σφάλματα και η ακεραιότητα των δεδομένων που έχουν αποθηκευτεί στο hdfs. Η πρώτη αντιμετωπίζεται με την έννοια της εφεδρικότητας (replication) των αποθηκευμένων block στο σύστημα. Συγκεκριμένα όταν επιχειρούμε να αποθηκεύσουμε ένα αρχείο στο σύστημα, αυτό εφ'όσον διαχωρίσει το συνολικό αρχείο σε επιμέρους blocks δημιουργεί εφεδρικά αντίγραφα (replicas) κάθε ενός από αυτά, και επιχειρεί να τα αποθηκεύσει σε διαφορετικούς κόμβους του cluster. Όπως είναι εμφανές αυτή η τακτική δίνει τη δυνατότητα στο σύστημα να επιλύει εύκολα πιθανά προβλήματα με συγκεκριμένους κόμβους αφού η ίδια πληροφορία είναι αποθηκευμένη και σε εναλλακτικούς κόμβους που μπορούν να εξυπηρετήσουν την ζήτηση. Η ακεραιότητα των δεδομένων από την άλλη, διασφαλίζεται διατηρώντας ένα checksum που αντιστοιχίζεται με κάθε αποθηκευμένο block και αποθηκεύεται και αυτό από το σύστημα. Όταν επιχειρείται ανάγνωση ενός συγκεκριμένου block ενός αρχείου γίνεται παράλληλα και έλεγχος του checksum. Αν ο έλεγχος αποτύχει τα replicas του συγκεκριμένου block αντικαθιστούν το αρχικό.

Το μοντέλο αρχιτεκτονικής συστήματος που χρησιμοποιεί το hdfs είναι αυτό του master/slave όπως φαίνεται και στο Σχήμα 2.2. Πιο αναλυτικά ένας κόμβος του cluster είναι ο λεγόμενος namenode (master) όπου εκεί εκτελείται και η διεργασία με το αντίστοιχο όνομα. Όλοι οι υπόλοιποι κόμβοι λέγονται datanodes (slaves) όπου επίσης εκτελούνται διεργασίες datanode. Η επικοινωνία των κόμβων γίνεται μέσω δικτύου με τα πρωτόκολλα TCP/IP και με RPC (Remote Procedure Calls) κλήσεις.




*Σχήμα 2.2: HDFS Architecture*

Ο namenode είναι ο επόπτης του συστήματος, με την έννοια ότι διαχειρίζεται το χώρο ονομάτων (namespace) του συστήματος αρχείων. Αυτό το καταφέρνει διατηρώντας το δέντρο του filesystem καθώς και μεταδεδομένα για όλα τα αρχεία και τους φακέλους σε

αυτό το δέντρο στο τοπικό του δίσκο. Ο namenode ενημερώνεται για οποιαδήποτε αλλαγή γίνεται στο filesystem και ανα πάσα στιγμή γνωρίζει την τοποθεσία κάθε block κάθε αρχείου στο cluster. Οι datanodes είναι αυτοί που παρέχουν τον κατανεμημένο χώρο αποθήκευσης και ο οποίος είναι ανεξάρτητος του τοπικού συστήματος αρχείου που παρέχει το κάθε λειτουργικό σύστημα. Είναι αυτοί που είναι υπεύθυνοι να αποθηκεύουν, να ανακαλούν και να διαγράφουν blocks δεδομένων από τον δίσκο τους όταν αυτό τους ζητηθεί, και να ανακοινώνουν περιοδικά στον namenode τα blocks που διατηρούν στη μνήμη τους.

Ανάμεσα στα καθήκοντα των datanodes είναι ακόμα να παρέχουν περιοδικά πληροφορία για την κατάσταση λειτουργίας τους με την μέθοδο του heartbeating στον namenode. Σε περίπτωση που κάποιος κόμβος αποτύχει επανειλημμένα τότε θεωρείται ότι βρίσκεται σε κατάσταση αδυναμίας λειτουργίας και δεν χρησιμοποείται πλέον για τις λειτουργίες του συστήματος μέχρι να επανέλθει. Αυτός είναι και ο αυτοματοποιημένος τρόπος εντοπισμού προβλημάτων που αναφέραμε στην αρχή της ενότητας. Επίσης στα πλαίσια της ανοχής σφαλμάτων που παρέχει το hdfs συνήθως λειτουργεί και ένας ακόμα κόμβος του cluster ως secondary namenode και παρέχει υποστήριξη σε περίπτωση βλάβης του πρωταρχικού namenode. Πιο ειδικά ο secondary namenode διατηρεί τακτικά ορόσημα (checkpoints) του δέντρου του filesystem που κανονικά διαχειρίζεται ο namenode, με αυτό το τρόπο είναι δυνατόν να αναλάβει τα καθήκοντά του συνήθως με το κόστος μια σημαντικής καθυστέρησης.

Το πιο ενδιαφέρον ίσως σημείο που αξίζει να αναλύσουμε διεξοδικότερα λόγω του ότι σχετίζεται και με την φύση της χρήσης του hdfs που θα κάνουμε στην παρούσα εργασία, είναι ο τρόπος που το σύστημα διαχειρίζεται μια αίτηση ανάγνωσης ενός αποθηκευμένου αρχείου στο filesystem. Όταν η εφαρμογή πελάτης του hdfs αιτείται ανάγνωσης ενός αρχείου από το filesystem, αρχικά επιδιώκει επικοινωνία με τον namenode κόμβο του cluster μέσω κλήσεων RPC. Ο namenode με τη σειρά του επιστρέφει στο πελάτη μια λίστα από διευθύνσεις κόμβων στο δίκτυο του cluster για κάθε ένα από τα block που συνθέτουν το ζητούμενο αρχείο. Οι διευθύνσεις αυτές είναι ταξινομημένες με τέτοια σειρά ώστε οι πρώτες να αντιστοιχίζονται σε κοντινότερες αποστάσεις κόμβων του cluster από τον κόμβο που εκτελεί τον κώδικα πελάτη. Αυτή η τελευταία λεπτομέρεια είναι ιδιαίτερα σημαντική στην όλη διαδικασία γιατί ουσιαστικά με αυτό τον τρόπο το σύστημα δηλώνει γνώση της τοπολογίας του cluster, κάτι που ονομάζεται rack awareness. Από τη στιγμή που ο κώδικας πελάτη έχει στη κατοχή του όλες τις διευθύνσεις των block του αρχείου είναι πλέον ελεύθερος να ζητήσει τα αντίστοιχα block από τους datanodes με τις συγκεκριμένες διευθύνσεις. Οποιοδήποτε σφάλμα κατά τη διάρκεια της ανάγνωσης από τους datanodes εμφανιστεί, είτε αυτό είναι πρόβλημα ακεραιότητας ενός συγκεκριμένου block είτε αδυναμία κάποιου datanode να διαχειριστεί αιτήσεις, επιλύεται άμεσα από τον κώδικα πελάτη αφού έχει εξ'αρχής στην κατοχή του τις εφεδρικές διευθύνσεις των block.

Όπως είναι φυσικό μια εφαρμογή που αποθηκεύει τα δεδομένα της σε ένα filesystem είτε είναι κατανεμημένο όπως το hdfs είτε κεντρικό, έχει την απαίτηση από αυτό να παρέχει τα δεδομένα με τον ταχύτερο δυνατό τρόπο και με τους απαραίτητους ελέγχους ακεραιότητας. Σε ένα κατανεμημένο περιβάλλον όπου η επικοινωνία γίνεται πάνω από δίκτυο οι απαιτήσεις αυτές γίνονται ακόμα πιο δύσκολα ικανοποιήσιμες και απαιτούν σύνθετο και έξυπνο σχεδιασμό από την πλευρά του συστήματος. Στη περίπτωση του hdfs αυτό επιτυγχάνεται με τα replicas των blocks και με την ιδιότητα του rack awareness που αναφέραμε στη προηγούμενη παράγραφο. Ειδικότερα αν αναλογιστούμε ότι πολλές

εφαρμογές που εκτελούνται χρησιμοποιώντας το hdfs τρέχουν τις υπηρεσίες τους στους ίδιους κόμβους με το hdfs cluster τότε η δυνατότητα να να διαβάζει ένας κόμβος ένα block τοπικά από τον δίσκο του είναι ιδιαίτερα αποδοτικό σε σχέση με το να αιτηθεί το ίδιο block να μεταφερθεί μέσω δικτύου από έταιρο κόμβο. Σε αυτές τις εφαρμογές η πληθικότητα που επιλέγεται για τα replicas μεταφράζεται ταυτόχρονα σε μεγαλύτερη ικανότητα του συστήματος να δρομολογήσει κατανεμημένες εργασίες στους ίδιους κόμβους όπου υπάρχουν και τα δεδομένα που θα ζητηθούν, και άρα να επωφεληθεί σε απόδοση.

## 2.2.3 Apache Spark

### 2.2.3.1 Εισαγωγή

Το Apache Spark είναι μια υπολογιστική πλατφόρμα ειδικά σχεδιασμένη για cluster υπολογιστών (cluster computing platform) υλοποιημένη στην γλώσσα προγραμματισμού Scala[10]. Είναι κατασκευασμένη ώστε να υποστηρίζει κατανεμημένες εφαρμογές γενικού σκοπού που βασίζονται εν γένει στην επεξεργασία μεγάλου όγκου δεδομένων, με μεγάλο βαθμό αποδοτικότητας και ταχύτητας.

Σε ένα γενικότερο πλαίσιο το Spark είναι προέκταση του προγραμματιστικού μοντέλου MapReduce, με τη κύρια διαφορά ότι υποστηρίζει περισσότερους τύπους υπολογισμών, όπως διαδραστικά ερωτήματα (interactive queries) και επεξεργασία δεδομένων συνεχούς ροής (streaming data processing). Μία ακόμα διαφορά σε σχέση με τις εργασίες (jobs) που μπορούν να ανατεθούν στο Spark σε σχέση με τις υλοποιήσεις του MapReduce σε άλλα συστήματα, είναι η ικανότητα που παρέχει για αποθήκευση δεδομένων στην μνήμη του κάθε κόμβου στο cluster κατά τη διάρκεια της εκτέλεσης του job. Αυτή η τελευταία ιδιότητα περιγράφεται ως caching και είναι ίσως ένα από τα πιο σημαντικά χαρακτηριστικά που εισάγει το Spark στην ανάπτυξη κατανεμημένων συστημάτων και του δίνει προβάδισμα στη ταχύτητα έναντι του Hadoop MapReduce πετυχαίνοντας μάλιστα μέχρι και 100 φορές καλύτερη χρονική απόδοση [11].

Η δομή του Spark περιγράφεται καλύτερα ως αυτή μιας ενιαίας στοίβας υποσυστημάτων που συνεργάζονται και παρέχουν το κάθε ένα από αυτά ξεχωριστές υπηρεσίες . Συγκεκριμένα η 'καρδιά' τους συστήματος είναι το Spark core. Το υποσύστημα core είναι η υπολογιστική μηχανή του συστήματος που είναι υπεύθυνη για την δρομολόγηση, τον διαμοιρασμό και την επίβλεψη των εφαρμογών οι οποίες αναλύονται σε επιμέρους υπολογιστικές μονάδες (tasks) και αναθέτονται σε κόμβους του cluster. Το core είναι επίσης αυτό που παρέχει τις διεπαφές για τα δομικά προγραμματιστικά στοιχεία του συστήματος όπως είναι για παράδειγμα τα Resilient Distributed Datasets (RDD) που θα αναλύσουμε αργότερα. Αποτελεί επίσης την βάση πάνω στην οποία στηρίζονται τα υποσυστήματα Spark Sql, Spark Streaming, MLib και GraphX. Συνοπτικά εδώ αναφέρουμε ότι το Spark Sql προσφέρει δυνατότητα για συνεργασία του Spark με δομημένα δεδομένα μέσω της γλώσσας Sql καθώς και της παραλλαγής αυτής που προσφέρει το Apache Hive. Το Spark Streaming είναι αυτό που δίνει την δυνατότητα αποθήκευσης και επεξεργασίας stream δεδομένων σε πραγματικό χρόνο. Το GraphX είναι μια βιβλιοθήκη ειδικά σχεδιασμένη για να παρέχει υποστήριξη για jobs και αλγορίθμους που επεξεργάζονται

γράφους. Τέλος η Mllib είναι η βιβλιοθήκη που διαθέτει συλλογή αλγορίθμων μηχανικής μάθησης υλοποιημένων με προσανατολισμό την κατανεμημένη εφαρμογή τους. Η Mllib είναι και αυτή που παρέχει την υλοποίηση του αλγορίθμου k-means στο Spark που θα μας απασχολήσει στη παρούσα εργασία.

Το Spark όπως προαναφέραμε είναι υλοποιημένο εξ'ολοκλήρου στη γλώσσα προγραμματισμού Scala, μια γλώσσα πολλαπλών παραδειγμάτων με ισχυρά στοιχεία συναρτησιακού αλλά και αντικειμενοστραφούς προγραμματισμού. Η scala χρησιμοποιεί το JVM περιβάλλον για την εκτέλεση των προγραμμάτων της, ενώ είναι σχεδιασμένη έτσι ώστε να μπορεί να εισάγει και να χρησιμοποιεί βιβλιοθήκες της Java. Όπως είναι φυσικό το Spark υποστηρίζει ανάπτυξη εφαρμογών σε οποιαδήποτε από τις γλώσσες Java και Scala ενώ εκτός αυτών υποστηρίζει και την γλώσσα Python. Τα παραπάνω καθιστούν το Spark ανεξάρτητο πλατφόρμας (platform independent) ενώ παράλληλα με την Python παρέχει μεγάλη ελευθερία ελευυερία επιλογής στους επίδοξους χρήστες του.

### 2.2.3.2 Αρχιτεκτονική του Spark

Η αρχιτεκτονική του Spark δεν αποτελεί εξαίρεση και ακολουθεί το αρχιτεκτονικό μοντέλο master/slave. Πιο ειδικά στη ορολογία του Spark υπάρχουν οι οντότητες του master και των workers. Όταν εκκινείται ένα Spark cluster μία διεργασία master εκτελείται στον κόμβο από όπου δίνεται η εντολή εκκίνησης, ενώ διεργασίες workers εκτελούνται στους κόμβους slaves.

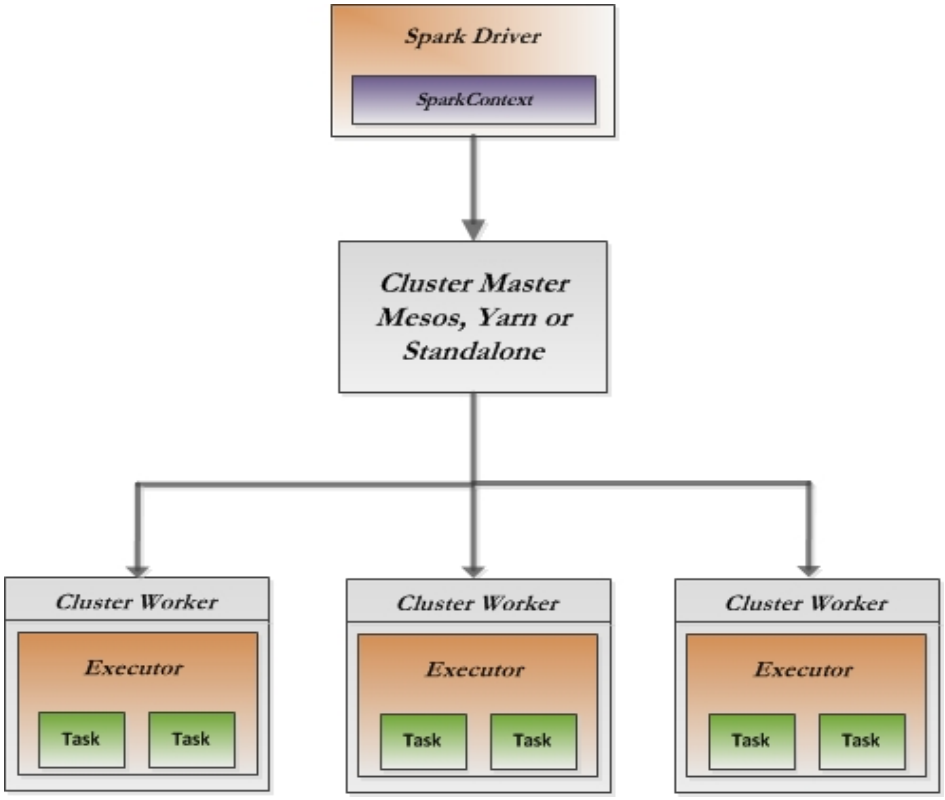
Εν γένει μια εφαρμογή που ξεκινάει να εκτελείται στο Spark χρειάζεται υπολογιστικούς πόρους (resources) από την πλευρά του cluster, που μεταφράζονται σε κύρια μνήμη και πυρήνες cpu (cpu cores). Ως επόπτη των διαθέσιμων resources του cluster το spark παρέχει ελευθερία επιλογής ανάμεσα στους Standalone,Yarn και Mesos cluster resource managers. Ο standalone που είναι και αυτός που θα χρησιμοποιήσουμε σε αυτή την εργασία και αποτελεί ενσωματωμένη υλοποίηση resource manager που παρέχει το spark. Οι διεργασίες λοιπόν του master και των worker αποτελούν κομμάτια του standalone resouce manager και εκκινούν έχοντας υπό την επίβλεψή τους ένα απόθεμα resources του κόμβου στον οποίον εκτελούνται. Εκτός αυτού διατηρείται συνεχής επικοινωνία μεταξύ των κόμβων workers και του κόμβου master με στόχο τον εντοπισμό τυχόν προβλημάτων. Τέλος οι διεργασίες αυτές εγκαθιστούν servers σε κάθε κόμβο ώστε να παρέχουν χρήσιμες πληροφορίες της λειτουργίας τους μέσω του πρωτοκόλλου http.

Οι δομικές μονάδες που συνιστούν μία εφαρμογή Spark κατά τη διάρκεια εκτέλεσής της είναι αυτές του driver και των executors όπως φαίνεται και στο Σχήμα 2.3:

**Executors**

Οι executors είναι οι java διεργασίες που εκκινούν οι workers σε κόμβους του cluster. Οι executor εκκινούνται στην αρχή κάθε job και η διάρκεια ζωή τους είναι όση και η διάρκεια ζωής του job. Οι executors είναι αυτοί που αναλαμβάνουν να κομμάτι του υπολογισμού που χρειάζεται για το job. Αυτό γίνεται με την ανάθεση ανεξάρτητων μονάδων υπολογισμού (tasks) σε κάθε executor. Συγκεκριμένα αποστολή κάθε executor είναι να δεσμεύσει τα απαραίτητα resources (μνήμη και cpu cores) και να εκτελέσει συγκεκριμένες

διαδικασίες που ορίζονται από τον κώδικα του job σε ένα κομμάτι (partition) των δεδομένων που επεξεργάζεται το job. Τα αποτελέσματα της δουλειάς τους διαφέρουν ανάλογα με το είδος του task. Αυτά είτε ανακοινώνονται πίσω στο driver που παρακολουθεί την διαδικασία εκτέλεσης, όπως θα δούμε, είτε αποθηκεύονται στη cache ή στο δίσκο ως ενδιάμεσα αποτελέσματα που θα καταναλωθούν από μελλοντικά task. Η αποθήκευση αυτή αναλαμβάνεται από μία υπηρεσία που ονομάζεται BlockManager και συνεργάζεται μαζί με τον executor αλλά και με τον driver. Τέλος είναι σημαντικό να τονίσουμε ότι η ανεξαρτησία των task ως υπολογιστικής μονάδας συνεπάγεται ότι όταν για κάποιο λόγο υπάρξει αποτυχία ολοκλήρωσης ενός task αυτό δεν επηρεάζει την εξέλιξη της εκτέλεσης του job παρά μόνο στο γεγονός ότι θα πρέπει να επανακινηθεί ένα πανομοιότυπο task.




*Σχήμα 2.3: Spark architecture*

## Driver

Ο driver είναι η διεργασία όπου βρίσκεται η main() μέθοδος της εφαρμογής του πελάτη. Σε αυτή υπάρχει το αντικείμενο SparkContext που αντιπροσωπεύει και ενθυλακώνει όλες τις επιλεγμένες ρυθμίσεις και παραμέτρους για ένα συγκεκριμένο job. Οι δύο βασικές υποχρεώσεις της διεργασίας driver συνοψίζονται ως εξής:

- *Μετατροπή της εφαρμογής σε επιμέρους tasks:*

Όταν ξεκινά να εκτελείται η εφαρμογή, ευθύνη του driver είναι να μετατρέψει τον κώδικα σε tasks τα οποία θα μπορέσουν να ανατεθούν στους executors που βρίσκονται διαθέσιμοι στους workers του cluster. Αυτή η διαδικασία ξεκινά με τον driver πρώτα να καταστρώνει ένα λογικό πλάνο εκτέλεσης του job (logical plan), το οποίο έχει την μορφή κατευθυνόμενου ακυκλικού γραφήματος, (Directed Acyclic Graph – DAG) των διαδικασιών που πρέπει να τελεστούν. Αφού καροριστεί το λογικό πλάνο εκτέλεσης πρέπει να μεταφραστεί σε φυσικό πλάνο εκτέλεσης. Στη φάση αυτή επιχειρούνται διάφορες βελτιστοποιήσεις στον τρόπο εκτέλεσης που μπορεί να πραγματοποιήσει ο driver. Τελικά μεταφράζει το λογικό πλάνο εκτέλεσης σε φυσικό πλάνο που αποτελείται από πολλαπλά tasks.

- *Δρομολόγηση (scheduling) tasks σε executors:*
  Η δρομολόγηση των tasks στους executors λαμβάνει χώρα ακριβώς μετά από την παραγωγή από τον driver του φυσικού πλάνου εκτέλεσης για το job. Σε αυτή τη φάση κάθε ένας executor έχει κάνει γνωστή την παρουσία του στο driver και αναμένει να του ανατεθούν task ώστε να ξεκινήσει κυρίως σώμα του υπολογισμού. Ο driver παρότι μπορεί να έχει πολλές δυνατές επιλογές για να αναθέσει τα task, κάνει έξυπνη επιλογή αναθέτοντάς όσο είναι δυνατό task στους κόμβους που έχουν αποθηκευμένα τοπικά και τα δεδομένα που θα χρειαστεί να επεξεργαστεί το εν λόγω task. Αυτά μπορεί να αφορούν είτε δεδομένα αποθηκευμένα στο τοπικό δίσκο του κόμβου και θα προσπελαστούν για πρώτη φορά είτε ακόμα και δεδομένα που έχουν αποθηκευτεί στη cache του κόμβου από προγενέστερο task που εκτελέστηκε εκεί. Σε οποιαδήποτε από τις δυο παραπάνω περιπτώσεις ο driver θα προτιμήσει πάντα το συγκεκριμένο κόμβο που πληρεί τα κριτήρια αυτά. Αυτό όπως είναι φυσικό δεν είναι πάντοτε εφικτό, αρκεί μόνο να σκεφτούμε την περίπτωση όπου υπάρχουν τέτοιοι κόμβοι στο cluster μας, αλλά τη συγκεκριμένη στιγμή είναι απασχολημένοι με την εκτέλεση ενός άλλου task. Παρόλα αυτά είναι τόσο περισσότερο προτιμητέα η επιλογή ενός τέτοιου κόμβου σε σχέση με έναν οποιοδήποτε άλλο, που το spark παρέχει και σχετική ρύθμιση (spark.locality.wait) που ορίζει τον αριθμό των δευτερολέπτων που μπορεί να αναμένει ο driver μέχρι να αποδεσμευτεί ένας κόμβος ώστε τελικά να του αναθέσει το task.

Όσα αναφέραμε παραπάνω αποτελούν τα δομικά στοιχεία της αρχιτεκτονικής του Spark αλλά και της αρχιτεκτονικής εκτέλεσης (runtime architecture). Εδώ θα συνοψίσουμε όσα ήδη αναφέραμε αναφέροντας τα βήματα που ακολουθούνται από την στιγμή που αρχίζει να εκτελείται η εφαρμογή πελάτη από τον Spark Driver:

1. Ο χρήστης υποβάλλει την εφαρμογή του με την βοήθεια ενός utility που προσφέρει το Spark και λέγεται spark-submit σε κάποιο κόμβο του cluster.
2. Το spark-submit με την σειρά του είναι υπεύθυνο να εκκινήσει τον driver στο κόμβο όπου και το ίδιο εκτελέστηκε.
3. Ο driver επικοινωνεί με τον master του cluster manager (π.χ Standalone) και ζητάει resources μνήμης και cpu cores που θα χρειαστούν οι executors για να εκτελεστεί το job.

4. Ο master αναθέτει στους workers να εκκινήσουν νέες διεργασίες executors με τα απαιτούμενα resources.
5. Οι executors δηλώνουν την επιτυχή εκκίνησή τους στον driver και αναμένουν tasks. Ο driver ορίζει tasks ανάλογα με το job που του έχει ανατεθεί, και τα δρομολογεί βέλτιστα στους executors του cluster.
6. Τα tasks εκτελούνται και είτε στέλνουν τα αποτελέσματά τους πίσω στον driver είτε αποθηκεύουν ενδιάμεσα την εργασία τους τοπικά (cache/δίσκος)
7. Όταν στην main() μέθοδο καλεστεί η μέθοδος stop() του αντικειμένου SparkContext, οι executors αποδεσμεύουν τα resources και τα αφήνουν εκ νέου στη διάθεση του cluster manager.

### 2.2.3.3 Resilient Distributed Datasets (RDD)

Ένα resilient distributed dataset είναι μια αμετάβλητη κατανεμημένη συλλογή αντικειμένων. Ουσιαστικά είναι μια δομή που καλύπτει με ένα επίπεδο αφαιρετικότητας (abstraction) τα προς επεξεργασία δεδομένα που χρησιμοποιούνται σε jobs του spark. Με αυτό τον τρόπο προσφέρεται από το σύστημα ένα σύνολο ιδιοτήτων και προγραμματιστικών διεπαφών στα δεδομένα, σε σχέση με αν αυτά τα μεταχειριζόμασταν στην φυσική τους υπόσταση (raw data). Το σύνολο των αντικειμένων μπορεί να αποτελείται από οποιοδήποτε τύπο δεδομένο υποστηρίζουν οι γλώσσες Python, Scala και Java είτε ακόμα και κλάσεις οριζόμενες από τον χρήστη.

Όλα τα jobs που υποβάλλονται στο Spark ξεκινούν ορίζοντας ένα RDD από μία πηγή δεδομένων (π.χ HDFS) ή από ένα ήδη υπάρχον RDD. Από τη στιγμή που ορίζεται ένα RDD αυτό πλέον ορίζει μια απεικόνιση των πραγματικών δεδομένων από τα οποία δημιουργήθηκε και διασπάται σε κομμάτια (partitions) που μπορούν να υπολογιστούν σε ξεχωριστούς κόμβους του cluster. Τα RDD είναι όπως αναφέραμε αμετάβλητα (immutable) με την έννοια ότι οι μόνες επιτρεπόμενες ενέργειες μετά τη δημιουργία τους είναι οι μετασχηματισμοί (transformations) και οι υπολογισμοί (actions).

Τα transformations είναι ουσιαστικά ενέργειες που επεξεργάζονται τα δεδομένα που ορίζονται από το RDD ανά partition, δημιουργώντας νέα RDD. Διαχωρίζονται όμως από τα actions γιατί λειτουργούν με οκνηρή αποτίμηση (lazy evaluation). Η έννοια του lazy evaluation, που δανείζεται από το τομέα των γλωσσών προγραμματισμού, υποδηλώνει ότι οποιοδήποτε transformation ορίζουμε σε ένα RDD δεν μετασχηματίζει την ίδια στιγμή το υποκείμενο dataset αλλά αντίθετα αποθηκεύονται για αυτό κάποια metadata που το χαρακτηρίζουν. Αυτό είναι ένα σημαντικό νέο χαρακτηριστικό που εισάγεται από το Spark και οφελεί πολύ στις περιπτώσεις όπου, από μια σειρά μετασχηματισμών που ορίζονται από ένα job και οδηγούν σε ένα αποτέλεσμα, μπορεί το ίδιο αποτέλεσμα να παραχθεί αποδοτικότερα χωρίς την σειριακή τους εκτέλεση.

Τα actions από την άλλη πλευρά είναι οι ενέργειες αυτές που εκτελούμε σε ένα RDD και υπολογίζουν ένα αποτέλεσμα. Για το υπολογισμό ενός αποτελέσματος χρειάζονται να γίνουν υπολογισμοί πάνω στα raw data που αντιπροσωπεύει το RDD, για αυτό και τα actions ουσιαστικά 'πυροδοτούν' μια σειρά μετασχηματισμών που ενδεχομένως έχουν οριστεί για ένα dataset και επιστρέφουν το ζητούμενο αποτέλεσμα είτε στον driver είτε το αποθηκεύουν κάπου (π.χ HDFS).

Ένα ακόμα σημαντικό πλεονέκτημα που παρουσιάζουν τα RDD είναι ο τρόπος που μοντελοποιούνται από το Spark ώστε να παρέχουν μηχανισμούς ανάκτησής τους σε περίπτωση προβλήματος. Συγκεκριμένα για κάθε RDD ανά πάσα στιγμή αποθηκεύεται η σειρά των μετασχηματισμών (lineage) που ακολουθήθηκαν για να κατασκευαστεί καθώς και πληροφορίες για την πηγή των δεδομένων που χρησιμοποιήθηκε για να κατασκευαστεί το συγκεκριμένο RDD. Αυτή η πληροφορία παρέχει την δυνατότητα στο Spark να επαναϋπολογίσει οποιοδήποτε partition του RDD χωρίς να επηρεάσει ουσιαστικά την εκτέλεση του job.

Τέλος είναι χρήσιμο να αναφέρουμε εδώ ότι το API των RDD δίνει τη δυνατότητα να κατασκευαστούν από πολλές διαφορετικές πηγές δεδομένων, κάτι που κάνει τον προγραμματισμό εφαρμογών στο Spark ιδιαίτερα απλή διαδικασία. Ανάμεσα στα άλλα παρέχεται και διεπαφή για κατασκευή RDD από το HDFS που ονομάζεται HadoopRDD. Ένα HadoopRDD στη πιο απλή του μορφή αντιστοιχίζει κάθε ένα block του αρχείου στο HDFS σε ένα partition του RDD, παρόλα αυτά το Spark παρέχει δυνατότητα στον προγραμματιστή να δημιουργήσει RDD από το HDFS με ρυθμιζόμενο αριθμό partitions. Το τελευταίο είναι κάτι που ανά περίσταση μπορεί να ωφελήσει την απόδοση ενός job.

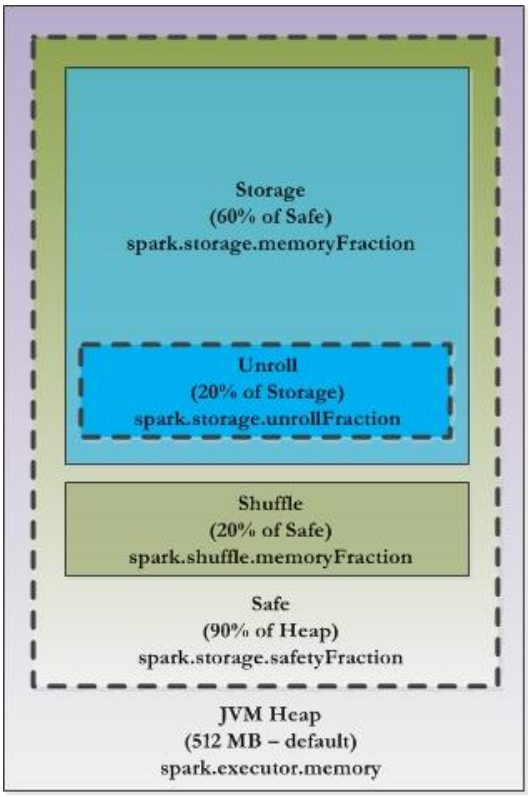
### 2.2.3.4 Caching

Όπως είδαμε μέχρι στιγμής το Apache Spark εισάγει στο χώρο των κατανεμημένων υπολογιστικών μηχανών κάποια πολύ ενδιαφέροντα στοιχεία, όπως τα RDD που αναλύσαμε στη προηγούμενη ενότητα. Αυτό όμως που εν γένει το κάνει πολύ ελκυστικό για εφαρμογές ανάλυσης δεδομένων (data analytics jobs) είναι η δυνατότητα που δίνει να αποθηκεύονται κατά τη διάρκεια εκτέλεσης ενός job ενδιάμεσα αποτελέσματα, (που είναι και αυτά με τη σειρά τους RDD) στην κύρια μνήμη των κόμβων του cluster, κάτι που ονομάζεται caching.

Παραδοσιακά υπολογιστικά μοντέλα όπως το MapReduce, που ακολουθούν παρόμοια φιλοσοφία εκτέλεσης όπως αυτή που περιγράψαμε στην ενότητα 2.3.2 μπορούνε μόνο να διατηρήσουν δεδομένα στη κύρια μνήμη στα πλαίσια που αυτά επεξεργάζονται από ένα συγκεκριμένο task κάποια δεδομένη στιγμή. Σε περίπτωση που τα ίδια δεδομένα χρειάζονται και από κάποιο επόμενο task που θα εκτελεστεί στον ίδιο κόμβο τότε αυτά παρέχονται από το δίσκο. Όπως είναι προφανές αυτό δημιουργεί ένα αδύναμο σημείο (bottleneck) στην εκτέλεση των task καθώς η ταχύτητα προσπέλασης δεδομένων στη κύρια μνήμη σε σύγκριση με τον δίσκο είναι υπερπολλαπλάσια. Πολύ σημαντικά οφέλη από το caching αποκομίζουν τα jobs εκείνα που χρησιμοποιούν επαναληπτικούς αλγόριθμους (iterative algorithms), που επεξεργάζονται σε κάθε επανάληψη το dataset με το οποίο τροφοδοτούνται. Ο k-means ανήκει σε αυτή τη κατηγορία αλγορίθμων και για αυτό κρίνουμε σκόπιμο στη παρούσα ενότητα να εξετάσουμε με λίγη περισσότερη λεπτομέρεια την έννοια του caching και πως αυτή υλοποιείται στο Spark.

Όπως έχουμε ήδη δει οι executors είναι οι java διεργασίες εκείνες που εκτελούνται σε κάθε κόμβο του cluster και πραγματοποιούν τον πυρήνα των υπολογισμών κάθε job. Καθότι είναι java διεργασίες εκτελούνται μέσα σε ένα Java Virtual Machine (JVM) το οποίο τους παρέχει κομμάτι μνήμης γνωστό ως heap, που είναι το διαθέσιμο περιβάλλον εργασίας της διεργασίας (workspace). Το heap χωρίζεται σε επι μέρους κομμάτια μνήμης τα οποία χρησιμεύουν για αποθήκευση διαφορετικού είδους δεδομένων που σχετίζονται με τη

διεργασία. Αυτός ο διαχωρισμός του heap απεικονίζεται από το Σχήμα 2.4 παρακάτω (σε default τιμές) μαζί με τις επιλογές που προσφέρει το spark για την ρύθμιση των μεγεθών τους.




*Σχήμα 2.4 : Executor's JVM heap*

Το συνολικό μέγεθος του heap είναι αυτό που εκφράζεται συνηθέστερα ως μία καθαρή τιμή ενώ όλα τα υπόλοιπα εκφράζονται ως ποσοστά αυτής της τιμής και αφορά το συνολικό workspace που διαθέτει το JVM στον executor. Συνοπτικά έχουμε:

- Safe region:
  Αυτή η περιοχή αποτελεί το 90% του συνολικού heap και ονομάζεται έτσι γιατί ακριβώς ορίζει ένα ανώτατο όριο το οποίο μπορούμε να δεσμεύσουμε από το heap ώστε να αποφύγουμε εξαιρέσεις ανεπαρκούς μνήμης (Out of Memory Error - OOM)

- Shuffle region:
  Η shuffle region καταλαμβάνει το 20% από την safe όπως φαίνεται και στο σχήμα και είναι σχεδιασμένη ώστε να αποθηκεύει δεδομένα τα οποία θα υποστούν shuffle. Η διαδικασία του shuffle λαμβάνει χώρα όταν δεδομένα που παράγει ένα task πρέπει να καταναλωθούν από ένα άλλο κόμβο που τρέχει ένα άλλο task στο cluster. Συνηθέστερα αυτά τα δεδομένα απαιτούνται να είναι και ταξινομημένα, οπότε το κομμάτι αυτό ικανοποιεί τις ανάγκες μνήμης για μια τέτοια ταξινόμηση.

- Storage region
  Το storage region που αποτελεί το 60% του safe region είναι αυτό που εξυπηρετεί την αποθήκευση των partitions ενός RDD σε ένα κόμβο. Από εδώ μπορούν να επαναπροσπελαστούν με πολύ μεγάλύτερη ταχύτητα από κάθε task.

- Unroll region
  Το spark παρέχει επίσης τη δυνατότητα να αποθηκεύονται τα δεδομένα σε σειροποιήσιμη μορφή (serialized) στη μνήμη ή στο δίσκο. Αυτή η μορφή δεδομένων απαιτείται από κατανεμημένα συστήματα όπως το Spark γιατί έτσι παρέχεται δυνατότητα μεταφοράς τους μέσω δικτύου. Στη serialized μορφή τα δεδομένα όμως δεν μπορούν να καταναλωθούν άμεσα οπότε χρειάζεται μια διαδικασία γνωστή ως deserialization. Τις ανάγκες αυτής της διαδικασίας σε μνήμη έρχεται να καλύψει η unroll region που αφορά το 20% της storage region.

Παρότι το caching είναι ισχυρός σύμμαχος της αποδοτικότητας ενός job αυτό δεν σημαίνει ότι το σύστημα δεν χρησιμοποιεί το δίσκο του κάθε κόμβου ως μέσο αποθήκευσης. Τουναντίον, όταν τα απαιτούμενα δεδομένα που επιλέγονται για caching ξεπερνούν τον διαθέσιμο χώρο που παρέχεται από τη storage region τότε αναπόφευκτα το Spark επιλέγει partitions τα οποία γράφει στο δίσκο. Ακόμα μπορεί να υπάρξει η ανάγκη για deserialization δεδομένων στην unroll region που να διεκδικήσει χώρο από τα αποθηκευμένα partitions όπου θα πρέπει να πράξει ανάλογα. Σε αυτές τις περιπτώσεις το Spark χρησιμοποιεί την πολιτική Least Recently Used (LRU) για να διαλέξει ποια partitions θα γράψει στο δίσκο. Σύμφωνα με αυτή τη πολιτική το πιο παλαιό χρονικά partition που έχει δεσμεύσει cache μνήμη διαλέγεται για αποχώρηση ώστε να αποθηκευτεί στη θέση του κάποιο νεότερο.

Στη εκτέλεση του αλγορίθμου k-means στη παρούσα διπλωματική επωφελούμαστε της επιλογής του caching καθώς όπως ήδη έχουμε περιγράψει η φύση του αλγορίθμου είναι τέτοια, που επαναεπεξεργάζεται σε κάθε βήμα όλο το dataset. Στο API της Python που χρησιμοποιήσαμε δίνεται η δυνατότητα στο προγραμματιστή να κατασκευάσει ένα StorageLevel object που το αρχικοποιεί με κατάλληλες παραμέτρους που υποδηλώνουν τη χρήση της μνήμης που θα κάνει η εφαρμογή κατά την εκτέλεσή της καθώς και την χρήση σειριοποιημένης αποθήκευσης ή μη του εκάστοτε RDD. Αυτό το αντικείμενο δίνεται ως παράμετρος στη μέθοδο persist() των RDD που μας ενδιαφέρουν. Η υπογραφή του αντικειμένου δίνεται εδώ:

*StorageLevel( useDisk, useMemory, useOfHeap, deserialized, replication=1 )*

Οι πρώτες τέσσερις παράμετροι δέχονται boolean τιμές True ή False ανάλογα με τη χρήση της μνήμης που θέλουμε να κάνουμε όπως αυτή υποδηλώνεται από το όνομα της παραμέτρου. Η τελευταία παράμετρος ( με default τιμή 1 ) υποδηλώνει το πόσα εφεδρικά αντίγραφα του συγκεκριμένου RDD ζητάμε να βρίσκονται στους κόμβους του cluster. Στη δική μας υλοποίηση θέλουμε χρήση όλης της διαθέσιμης μνήμης των κόμβων καθώς και χρήση του δίσκου εν ανάγκη, ενώ επίσης θέλουμε το dataset μας να διατηρείται serialized

στη μνήμη για αποδοτικότερη χρήση της. Με δεδομένα τα παραπάνω το StorageLevel object που χρησιμοποιούμε στο κώδικά μας θα είναι:

*StorageLevel( True, True, False, False, replication=1 )*

### 2.2.3.5 MLlib

Όπως αναφέρθηκε στην ενότητα 2.3.1 η MLlib είναι ένα από τα υποσυστήματα που εμπεριέχεται στη στοίβα λογισμικού που αποτελούν το Apache Spark. Ουσιαστικά παρέχει μια συλλογή αλγορίθμων machine learning και data science ειδικά σχεδιασμένων για την εκτέλεσή τους στο κατανεμημένο περιβάλλον πάνω από το Spark Core υποσύστημα. Εκτός αυτού ορίζεται μια συλλογή τύπων δεδομένων (data structures) όπως είναι τα labeled points και τα vectors που είναι ειδικά σχεδιασμένοι για τις ανάγκες αυτών των αλγορίθμων. Η ευκολία που παρέχει η MLlib είναι ότι όλα τα παραπάνω υλοποιούνται ως προγραμματιστικές διεπαφές των RDD, δηλαδή από τη προγραμματιστική πλευρά αρκεί η κλήση μιας μεθόδου σε ένα RDD για να γίνει χρήση των δυνατοτήτων της MLlib.

Όσο αφορά στον αλγόριθμο k-means που θα χρησιμοποιήσουμε στη παρούσα εργασία, για την εκτέλεσή του με την Mllib αρκεί να ορίσουμε ως RDD το dataset που θέλουμε να χρησιμοποιήσουμε για clustering και να καλέσουμε μία μέθοδο train() που δέχεται τα ορίσματα initializationMode, maxIterations και runs. Η επιλογή του initializationMode είναι αυτή που κάνει χρήση παραλλαγών του αλγορίθμου όπου γίνεται αρχική επιλογή κεντροϊδών με βελτιστοποιημένο τρόπο, αλλά παρέχεται και η δυνατότητα τυχαίας επιλογής με την παράμετρο random. Η παράμετρος maxIterations αφορά στο μέγιστο αριθμό επαναλήψεων που επιτρέπουμε στον αλγόριθμο να εκτελέσει σε περίπτωση που δεν επιτύχει σύγκλιση νωρίτερα. Τέλος η παράμετρος runs παρέχει τη δυνατότητα να εκτελεστούν παράλληλα περισσότερες της μίας εκτελέσεις του αλγορίθμου ξεκινώντας με διαφορετικά κεντροειδή, έτσι ώστε τελικό αποτέλεσμα επιλέγεται εκείνο που επέτυχε και το καλύτερο clustering.

## 2.2.4 Weka

Το Weka (Waikato Environment for Knowledge Analysis) είναι μία σουίτα λογισμικού ανοιχτού κώδικα που αναπτύχθηκε από το University of Waikato της Νέας Ζηλανδίας [12]. Παρέχει ένα εύρος αλγορίθμων που συγκαταλέγονται στην κατηγορία της μηχανικής μάθησης και χρησιμοποιούνται για προεπεξεργασία δεδομένων (data preprocessing), κατηγοριοποίηση (classification), παρεμβολή (regression), συσταδοποίηση (clustering) και οπτικοποίηση δεδομένων (data visualization). Είναι ιδιαίτερα χρήσιμο στους τομείς της ανάλυσης δεδομένων (data analysis) και της προγνωστικής μοντελοποίησης (predictive modeling). Είναι εξ'ολοκλήρου υλοποιημένο στη γλώσσα προγραμματισμού Java που το καθιστά αυτόματα platform independent ενώ διαθέτει τη δυνατότητα να χρησιμοποιηθεί είτε ως βιβλιοθήκη λογισμικού που ενσωματώνεται σε προγράμματα είτε από την γραφική διεπαφή (GUI) ως ανεξάρτητη εφαρμογή.

Στα πλαίσια της παρούσας διπλωματικής το Weka το χρησιμοποιούμε τόσο για την υλοποίηση του αλγορίθμου k-means σε κεντρικό περιβάλλον όσο και για την κατασκευή των predictive models που επιδιώκουμε να φτιάξουμε. Στη πρώτη περίπτωση το Weka παρέχει την κλάση SimpleKMeans που αποτελεί μέρος του πακέτου weka.clusterers η οποία παρέχει μεθόδους με τις οποίες ορίζουμε το αριθμό των clusters καθώς και τον μέγιστο αριθμό επαναλήψεων που θέλουμε να εκτελέσει ο αλγόριθμος. Το dataset που δέχεται ως είσοδο ο αλγόριθμος μπορεί να είναι είτε ένα csv αρχείο είτε ένα arff αρχείο με τα data points. Τα arff αρχεία αποτελούν ένα ειδικό format αρχείου που χρησιμοποιεί το weka. Όσο αναφορά τα predictive models αυτά χρησιμοποιούνται έυκολα από το GUI του Weka. Εκεί επιλέγουμε να φορτώσουμε ένα training set και ένα test set για το μοντέλο που θα κατασκευάσουμε τα οποία επίσης μπορούν είναι σε μία από τις μορφές που προαναφέραμε. Μετέπειτα διαλέγουμε έναν classifier (π.χ. Multilayer Perceptron) τον οποίο θέλουμε και το Weka αναλαμβάνει την εκπαίδευση του μοντέλου με το training set που έχουμε φορτώσει αλλά και την επαλήθευση του μοντέλου με το test set. Στο τέλος αναφέρει στατιστικά που έχουν σχέση με την ποιότητα του μοντέλου ( π.χ. μέση τιμή απόλυτου σφάλματος προβλέψεων) καθώς και οπτικοποίηση της επίδοσης του μοντέλου στο παρεχόμενο test set.

## 2.3 Μοντέλα και προβλεπτική μοντελοποίηση

Μακροσκοπικά ένα μοντέλο αποτελεί μια απλουστευμένη αναπαράσταση μιας διαδικασίας ή ενός συστήματος που παρατηρούμε και μελετούμε τη συμπεριφορά του στο φυσικό κόσμο. Αφαιρεί μεγάλη ποσότητα πληροφορίας και επικεντρώνεται στα χαρακτηριστικά εκείνα τα οποία είτε είναι τα πιο σημαντικά είτε τα πιο διαχειρίσιμα για το σκοπό της μελέτης.

Στο τομέα του data science και του machine learning ένα μοντέλο είναι ουσιαστικά μία φόρμουλα η οποία σκοπό έχει να εκτιμήσει ή να προβλέψει μια τιμή ενδιαφέροντος δεδομένου κάποιων τιμών εισόδου. Οι τιμές εισόδου αναπαριστούν κάποια ποσοτικοποιημένα χαρακτηριστικά που εκφράζουν την φυσική διαδικασία που επιχειρεί να αποδόσει το μοντέλο, ενώ η τιμή εξόδου αναπαριστά την απόκρισή της για τα δεδομένα εισόδου. Συνηθέστερα η ανάπτυξη τέτοιων μοντέλων ακολουθεί τα παρακάτω βήματα:

- Συλλογή ενός συνόλου τιμών εισόδου και των αντίστοιχων τιμών εξόδου από το σύστημα όπως αυτές προκύπτουν από την παρατήρησή του
- Τροφοδότηση ενός ποσοστού του παραπάνω συνόλου με σκοπό την εκπαίδευση του μοντέλου και την εξεύρεση μαθηματικής συσχέτισης των δεδομένων εισόδου με την απόκριση του συστήματος
- Τροφοδότηση του εναπομείναντος ποσοστού στο μοντέλο για αξιολόγηση της απόδοσή του

Στη παρούσα εργασία τα υπό παρατήρηση συστήματα είναι οι μηχανές εκτέλεσης Weka και Apache Spark ενώ οι τιμές εξόδου αυτών είναι η συλλογή των μετρικών απόδοσης και της χρονικής διάρκειας εκτέλεσης για τιμές εισόδου που αφορούν τις διαφορετικές παραμέτρους εκτέλεσης του αλγορίθμου k-means που επιλέγουμε κάθε φορά.

Υπάρχει μια πληθώρα μοντέλων που έχουν προταθεί και αναπτυχθεί και εκτείνονται από απλές μαθηματικές διαδικασίες όπως η απλή γραμμική παρεμβολή (simple linear regression), τα δέντρα απόφασης (decision trees) μέχρι νευρωνικά δίκτυα. Αναφέρουμε εδώ συνοπτικά λίγα θεωρητικά στοιχεία για κάθε ένα από τα προαναφερθέντα τα οποία ανά περίπτωση μετρικής και μηχανής εκτέλεσης χρησιμοποίησαμε στη μελέτη που θα ακολουθήσει.

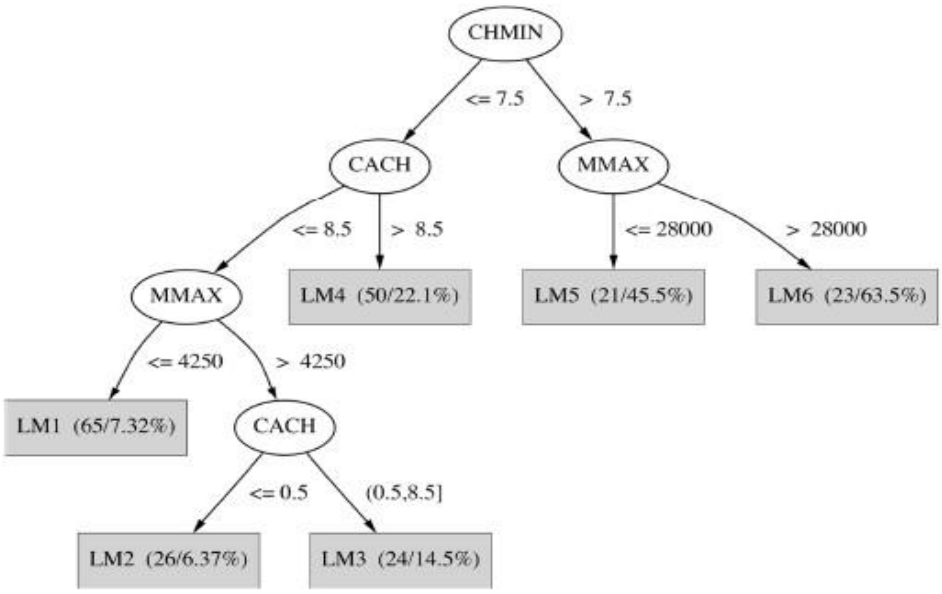
- **Γραμμική Παρεμβολή (linear regression)**
  Στη περίπτωση των linear regression μοντέλων γίνεται η παραδοχή της γραμμικής εξάρτησης των μεταβλητών εισόδου του συστήματος (ανεξάρτητες μεταβλητές) σε σχέση με τη μεταβλητή εξόδου (εξαρτημένη μεταβλητή) σε μία σχέση της μορφής που φαίνεται παρακάτω:

$$y_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i$$
$$i = 1, \ldots, n$$

  Στη παραπάνω σχέση (εκράζεται και διανυσματικά) η τιμή της μεταβλητής y είναι αυτή που προσδοκούμε να εκτιμήσουμε/προβλέψουμε ενώ οι μεταβλητές x είναι οι ν-πλήθους τιμές εισόδου που εισάγουμε εμείς στο σύστημα. Η τιμή ε εκφράζει μια τυχαία μεταβλητή που ανταποκρίνεται στο σφάλμα των μετρήσεών μας και εκφράζει μαθηματικώς την στοχαστικότητα του περιβάλλοντος. Η εκπαίδευση ενός linear regression μοντέλου με βάση τα συλλεχθέντα δεδομένα, εναπόκειται στην εύρεση και εκτίμηση των παραγόντων β της παραπάνω γραμμικής σχέσης. Η προσέγγιση των τιμών των παραγόντων β γίνεται με κριτήριο την ελαχιστοποίηση του αθροίσματος των τετραγωνισμένων διαφορών των παρατηρήσιμων τιμών της μεταβλητής ενδιαφέροντος και των αντίστοιχων εκτιμώμενων τιμών. Για το σκοπό αυτό έχουν προταθεί μία σειρά μεθόδων εκ των οποίων μία από τις πιο γνωστές είναι η μέθοδος ελαχίστων τετραγώνων[13].

- **M5P model trees**
  Τα M5P model trees αποτελούν μία υποπερίπτωση των κλασικών δεντρικών μοντέλων απόφασης. Μακροσκοπικά σε αυτά τα δέντρα απόφασης κάθε κόμβος, εκκινόντας από τη ρίζα, εμπεριέχει ένα κριτήριο διαχωρισμού (splitting criterion) με βάση τις τιμές μιας τιμής εισόδου (attribute) του συστήματος που παρατηρούμε. Με αυτό το κριτήριο δρομολογούμαστε διαμέσου όλων των ενδιάμεσων κόμβων από τη κορυφή μέχρι τα φύλλα του δέντρου απόφασης. Τα κλασσικά δέντρα παρεμβολή (regression trees) καταλήγουν στα φύλλα τους με μία εκτιμώμενη αριθμητική τιμή για την μεταβλητή εξόδου. Εν αντιθέσει στα model trees στα φύλλα του δέντρου κατασκευάζονται γραμμικά μοντέλα, όπως αυτά που είδαμε προηγουμένως, με το σύνολο των παρατηρήσιμων τιμών με τις οποίες δρομολογούμαστε στα εν λόγω φύλλα. Το κριτήριο διαχωρισμού στη περίπτωση των M5P model trees για κάθε κόμβο είναι το ονομαζόμενο SDR (standard deviation reduction)[14], σύμφωνα με το οποίο σε κάθε ενδιάμεσο κόμβο απόφασης επιλέγουμε να κάνουμε διαχωρισμό του συνόλου των παρατηρήσιμων

τιμών του συστήματος σε υποσύνολα, βάσει της τιμής μιας μεταβλητής εισόδου με σκοπό την μέγιστη μείωση της τυπικής απόκλισης (standard deviation) που επιτυγχάνεται από τον διαχωρισμό αυτό. Τη κατασκευή αυτών των μοντέλων αναλαμβάνουν tree induction αλγόριθμοι οι οποίοι τερματίζουν όταν επιπρόσθετοι διαχωρισμοί οδηγούν σε μη σημαντική μείωση του SDR. Ένα παράδειγμα M5P tree model φαίνεται στο Σχήμα 2.5 για κάποια attributes ενός συστήματος. Σε κάθε κόμβο φαίνονται οι τιμές διαχωρισμού που βρέθηκαν από την ικανοποίηση του SDR splitting criterion.




*Σχήμα 2.5: Παράδειγμα M5P tree model*

- **Multilayer Perceptron**

  Το multilayer perceptron είναι ένα εμπρόσθια τροφοδοτούμενο νευρωνικό δίκτυο το οποίο αποτελείται στη γενική περίπτωση από ένα ή και περισσότερα επίπεδα νευρώνων. Κάθε νευρώνας είναι εφοδιασμένος με μία συνάρτηση ενεργοποίησης (activation function) που ως είσοδο δέχεται το άθροισμα των γινομένων των εξόδων του προηγούμενου επιπέδου και των βαρών (weights) που αντιστοιχούν σε κάθε μία από αυτές. Ως είσοδος στο πρώτο επίπεδο του νευρωνικού δικτύου είναι οι μεταβλητές εισόδου του συστήματος που παρατηρούμε ενώ έξοδος είναι η εκτιμώμενη τιμή της μεταβλητής ενδιαφέροντος. Η εκπαίδευση του νευρωνικού δικτύου συνίσταται στην μεταβολή των βαρών για κάθε επίπεδο και για κάθε είσοδο νευρώνα σύμφωνα με τον αλγόριθμο back propagation [15] με ένα σύνολο εκπαίδευσης (training set) που αφορά ένα ποσοστό των παρατηρούμενων μετρήσεων του συστήματος.

# Κεφάλαιο 3

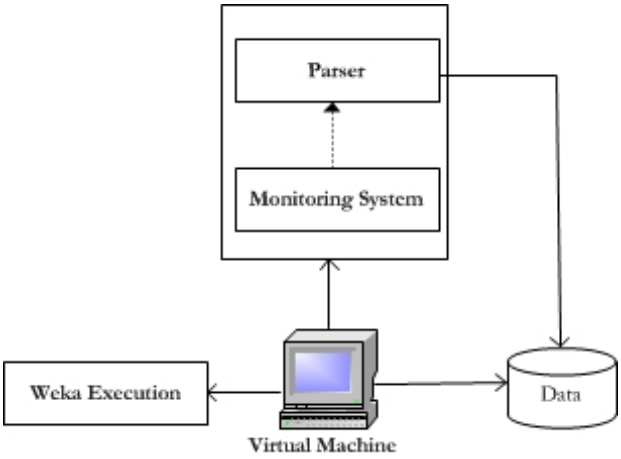
# Αρχιτεκτονική και εργαλεία συστήματος Profiling

Σκοπός του κεφαλαίου αυτού είναι να περιγραφεί και να αναλυθεί τόσο η αρχιτεκτονική (κεντρική και κατανεμημένη) του συστήματος profiling, καθώς και τα εργαλεία που χρησιμοποιήθηκαν στη παρούσα διπλωματική για την ανάκτηση μετρικών απόδοσης από την εκτέλεση του αλγορίθμου k-means στη τελική τους μορφή. Ξεκινάμε περιγράφοντας συνοπτικά την αρχιτεκτονική μας για την κεντρική και κατανεμημένη εκτέλεση. Συνεχίζουμε αναλύοντας το Ganglia monitoring system και τα εργαλεία με τα οποίο συγκεντρώσαμε τις μη μορφοποιημένες μετρήσεις για κάθε εκτέλεση του αλγορίθμου. Ολοκληρώνουμε το κεφάλαιο περιγράφοντας την διαδικασία parsing που ακολουθήθηκε για την τελική μορφή των δεδομένων της διαδικασίας profiling, καθώς και τον Openstack cloud όπου χρησιμοποιήθηκε ως περιβάλλον ανάπτυξης όλων μας των εργασιών.

## 3.1 Περιγραφή αρχιτεκτονικής και διαδικασίας profiling

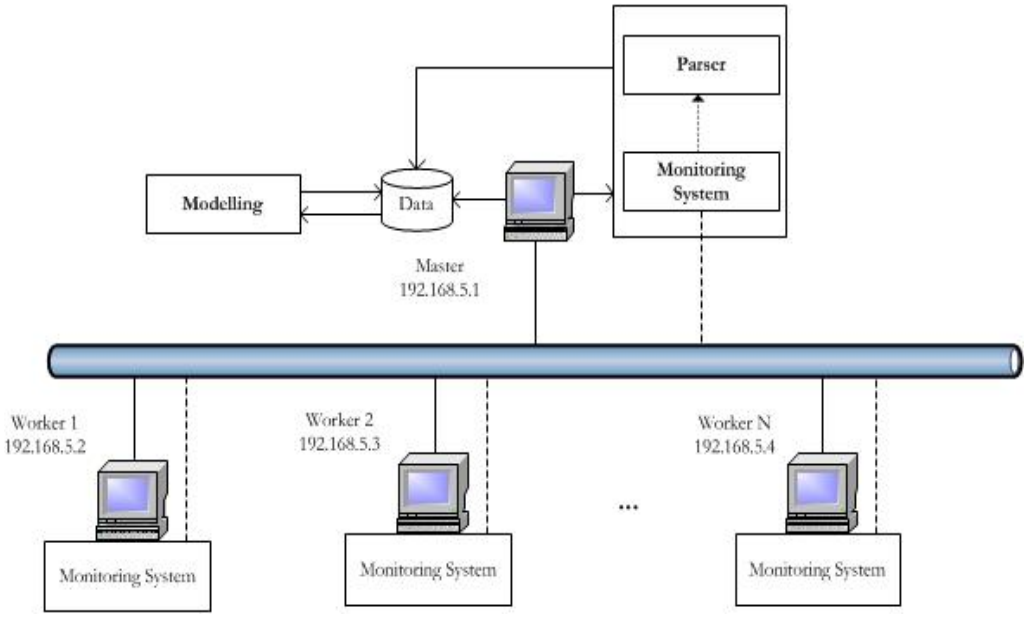
### 3.1.1 Κεντρική και κατανεμημένη αρχιτεκτονική

Για την υλοποίηση του αλγορίθμου k-means και την διαδικασία profiling χρησιμοποιήθηκε μια κεντρική αρχιτεκτονική που αφορά στην υλοποίηση του σε weka, και μία κατανεμημένη που αφορά στην υλοποίηση του στο apache spark. Και οι δυο παραπάνω φαίνονται γραφικά στα Σχήματα 3.1 και 3.2 που ακολουθούν.




*Σχήμα 3.1: Κεντρική αρχιτεκτονική συστήματος profiling*

Όπως φαίνεται και από το Σχήμα 3.1, στη κεντρική υλοποίηση έχουμε μόνο έναν κόμβο ο οποίος είναι ουσιαστικά ένα virtual machine που κατασκευάσαμε στο cloud για τους σκοπούς της μελέτης μας. Αυτό το virtual machine ενσωματώνει την λογική εκτέλεσης μαζί με το σύστημα monitoring και τον parser που θα αναλυθούν εκτενέστερα στις επόμενες ενότητες. Η διαδικασία σε αφαιρετικό επίπεδο περιλαμβάνει την ενεργοποίηση του συστήματος monitoring στο κόμβο το οποίο συλλέγει μετρικές απόδοσης (π.χ χρήση κύριας μνήμης), ενώ επίσης υπολογίζει και το συνολικό χρόνο εκτέλεσης. Μετά το πέρας της εκτέλεσης τα δεδομένα είναι διαθέσιμα για τον parser που αναλαμβάνει την μορφοποίησή τους και την τελικά αποθήκευση στο τοπικό μέσο αποθήκευσης. Η τελική μορφή των δεδομένων είναι αυτή που είναι κατάλληλη για την τροφοδότηση στα μοντέλα που θέλουμε να εκπαιδεύσουμε.




*Σχήμα 3.2: Κατανεμημένη αρχιτεκτονική συστήματος profiling*

Στη κατανεμημένη αρχιτεκτονική του Σχήματος 3.2 οι κόμβοι χωρίζονται σε N workers και ένα master σύμφωνα με αυτά που ήδη έχουμε αναφέρει για το σύστημα Spark

και γίνεται σύνδεση όλων των κόμβων σε ένα δίκτυο Ethernet. Αυτοί οι κόμβοι μαζί με το master τρέχουν τις απαραίτητες διεργασίες worker και executor του Spark καθώς και τις διεργασίες Datanode και Namenode, οι οποίες δεν σημειώθηκαν στο σχήμα χάριν εξοικονόμησης χώρου. Η διαδικασία όπως περιγράφηκε παραπάνω για την κεντρική αρχιτεκτονική παραμένει κατουσία ίδια, με τη κυριότερη διαφορά ότι το monitoring system λειτουργεί κατανεμημένα με στόχο την συλλογή μετρικών από κάθε κόμβο που συμμετέχει στο cluster. Η επικοινωνία των μετρικών γίνεται πάνω από το δίκτυο με τρόπο που αναλαμβάνει το Ganglia όπως θα εξηγηθεί στην επόμενη ενότητα. Η συλλογή όλων των μετρικών γίνεται από τον κόμβο master όπου από εκεί χρησιμοποιούνται από τον parser για την τελική τους αποθήκευση στο τοπικό χώρο αποθήκευσης του master. Τα αποτελέσματα του parser είναι επίσης εκείνα που τροφοδοτούνται για τη διαδικασία του modelling την οποία θα παρουσιάσουμε σε επόμενες ενότητες.

## 3.2 Εργαλεία monitoring

### 3.2.1 Ganglia monitoring system

Για το κομμάτι της αρχιτεκτονικής profiling που υλοποιεί το monitoring που είδαμε στη προηγούμενη ενότητα, χρησιμοποιήσαμε το Ganglia Monitoring System [16]. Το Ganglia είναι ένα κατανεμημένο σύστημα που εγκαθίσταται σε cluster ή ακόμα και σε grid που αποτελούνται από cluster υπολογιστών, και αναλαμβάνει την συλλογή μιας πληθώρας μετρικών που σχετίζονται με την χρήση των υπολογιστικών πόρων (π.χ cpu load) κάθε κόμβου κάθε στιγμή, αλλά και την αναφορά στατικών χαρακτηριστικών του (π.χ τύπος operating system). Παρότι ο σχεδιασμός του είναι προορισμένος για κατανεμημένες τοπολογίες μπορεί εξίσου αποδοτικά να χρησιμοποιηθεί και για έναν μόνον κόμβο. Για αυτό το λόγο το Ganglia χρησιμοποιήθηκε και για την κεντρική αλλά και για την κατανεμημένη αρχιτεκτονική που είδαμε προηγουμένως.

Η γενική φιλοσοφία λειτουργίας του Ganglia στηρίζεται στην επικοινωνία όλων των κόμβων για τους οποίους είναι υπεύθυνο να προσφέρει υπηρεσίες monitoring σε ένα multicast κανάλι επικοινωνίας όπου κάθε κόμβος αναμεταδίδει μετρικές που έχουν σχέση με τον ευατό του ενώ παράλληλα εισακούει τα αντίστοιχα δεδομένα από τους υπόλοιπους κόμβους. Η παραπάνω επικοινωνία γίνεται μέσω ενός απλού πρωτοκόλλου ακοής/ανακοίνωσης (listen/announce protocol) μέσω XDR (External Data Representation) [17]. Οι μετρικές για όλο το Ganglia cluster συγκεντρώνονται περιοδικά από έναν poller που τα αποθηκεύει σε κατάλληλη βάση δεδομένων από όπου μπορούν να ανακτηθούν. Παράλληλα παρέχεται η δυνατότητα οπτικοποίησης των δεδομένων αυτών μέσω Web UI που μπορεί να υποστηρίζει το Ganglia μέσω του υποσυστήματος gweb και με τη βοήθεια ενός apache web server [18].

Η παραπάνω περιγραφείσα λειτουργία στηρίζεται σε 2 τύπους δαιμόνων/υπηρεσιών του Ganglia ( daemons ) καθώς και μίας ειδικά σχεδιασμένης round robin database που χρησιμοποιεί για την αποθήκευση των μετρικών που συλλέγονται. Συνοπτικά αναλύουμε τα δομικά αυτά στοιχεία εδώ:

- **Gmond :** Ο ganglia monitoring daemon (gmond) είναι ο daemon που εγκαθίσταται σε κάθε κόμβο του cluster που μας ενδιαφέρει η συλλογή μετρικών. Λειτουργεί πλήρως ανεξάρτητα από το υπόλοιπο σύστημα και ευθύνη του είναι η συλλογή των απαιτούμενων μετρικών του συστήματος σε συνεργασία με το λειτουργικό σύστημα του κόμβου και η ανακοίνωσή τους στο multicast κανάλι (μέσω πρωτοκόλλου udp) που συμμετέχουν όλοι οι κόμβοι και ανακοινώνουν τις μετρήσεις τους. Η συλλογή αυτών των μετρικών γίνεται σε οριζόμενα από τον χρήστη χρονικά διαστήματα όπως αυτά ορίζονται μέσω ενός configuration file. Επίσης υπ'ευθύνη του gmond γίνεται και η συλλογή των μετρικών που ανακοινώνονται στο παραπάνω κανάλι από τους υπόλοιπους κόμβους. Η παραπάνω λειτουργία καθιστά κάθε κόμβο ενήμερο για τις μετρικές όλου του cluster ανά πάσα στιγμή. Το τελευταίο είναι πολύ σημαντικό γιατί λειτουργεί ως ασφάλεια των δεδομένων αφού αυτά είναι διαθέσιμα από κάθε κόμβο στο cluster.

- **Gmetad :** Ο ganglia meta daemon (gmetad) είναι ο daemon που συλλέγει και συσσωρεύει τις μετρικές από ένα gmond daemon του cluster που του παρέχει το σύνολο των μετρικών για όλα τα μηχανήματα του cluster τη στιγμή εκείνη. Τις μετρικές αυτές τις αποθηκεύει σε rrd αρχεία που αποτελούν ειδικά αρχεία που διαχειρίζεται η rrd βάση δεδομένων και κατηγοριοποιούνται ανά κόμβο και ανά μέτρηση σε φάκελο του filesystem του μηχανήματος στο οποίο έχει εκκινήσει ο gmetad.

- **RRD Database :** Η round robin database είναι το αποθηκευτικό σύστημα που χρησιμοποιεί το ganglia για να διαχειρίζεται τις συλλεγόμενες μετρικές. Η rrd καταφέρνει να αποθηκεύει μεγάλο όγκο δεδομένων διατηρώντας τις πιο πρόσφατες μετρήσεις ακριβείς και συμπυκνώνοντας τις παλαιότερες συλλέγοντάς τις και υπολογίζοντας το μέσο όρο τους σε τακτά χρονικά σημεία. Η rrd εγκαθίσταται στον κόμβο όπου βρίσκεται και ο gmetad daemon με τον οποίο συνεργάζεται. Εκτός από πρακτικό μέσο αποθήκευσης των μετρικών η rrd έχει και visualization προεκτάσεις. Συγκεκριμένα μπορεί να αναλάβει να οπτικοποιήσει τα δεδομένα που αποθηκεύει για μετρικές σε συγκεκριμένα format ως προς το χρόνο.

Οι μετρικές που συλλέγονται από το gmond daemon από κάθε κόμβο παραμετροποιούνται από το χρήστη ανάλογα με τις ανάγκες του, φορτώνοντας συγκεκριμένα modules. Για κάθε μετρική διατίθεται ένα αντίστοιχο module το οποίο είναι κομμάτι κώδικα του daemon που λειτουργεί ανεξάρτητα, επικοινωνώντας με το εκάστοτε λειτουργικό σύστημα από το οποίο συλλέγει τις μετρικές. Τα default modules με τα οποία λειτουργούν οι daemons gmond κατά την εγκατάσταση, επαρκούσαν για τις ανάγκες της παρούσας διπλωματικής, οπότε δεν χρησιμοποιήσαμε περαιτέρω modules. Οι μετρικές που χρησιμοποιήσαμε με το όνομα που αναφέρονται στο Ganglia, καθώς και μία σύνοπτική περιγραφή τους φαίνονται στον Πίνακα 3.2 παρακάτω:

| Όνομα Μετρικής | Περιγραφή |
|---|---|
| **bytes_in** | μέτρηση εισερχόμενης κίνησης δικτύου (KB) |
| **bytes_out** | μέτρηση εξερχόμενης κίνησης δικτύου (KB) |
| **mem_buffers** | buffers κύριας μνήμης (KB) |
| **mem_cached** | χρήση κύριας μνήμης ως cache |

| | |
|---|---|
| **mem_free** | διαθέσιμη κύρια μνήμη |
| **cpu_idle** | ποσοστό χρόνου που αντιστοιχεί σε αδράνεια του επεξεργαστή |
| **cpu_system** | ποσοστό χρόνου επεξεργαστή για εξυπηρέτηση κλήσεων συστήματος του λειτουργικού |
| **cpu_wio** | ποσοστό χρόνου επεξεργαστή που αναλίσκεται σε αναμονή για επικοινωνία με το δίσκο |
| **cpu_user** | ποσοστό χρόνου επεξεργαστή που αφιερώνεται για κώδικα περιβάλλοντος χρήστη (user space) |

*Πίνακας 3.1: Μετρικές Ganglia*

Εκτός των άλλων το Ganglia διαθέτει και το gweb Web UI που αναφέραμε στην αρχή. Το gweb είναι γραμμένο στη γλώσσα PHP και συνεργάζεται με την rrd καθώς και με ένα apache web server για να διαθέτει μέσω ίντερνετ visualizations των μετρικών ως προς το χρόνο για κάθε κόμβο. Ένα ακόμα σημαντικό στοιχείο του gweb που χρησιμοποιήθηκε για την εργασία μας είναι το events api. Το events api είναι μια διεπαφή που προσφέρει το gweb μέσω μεθόδων post του πρωτοκόλλου http για να την αποθήκευση δεδομένων σχετικά με την έναρξη και την λήξη γεγονότων ενδιαφέροντος. Συγκεκριμένα με αυτό τον τρόπο μπορέσαμε αυτοματοποιημένα να ενημερώνουμε το Ganglia για τις χρονικές στιγμές έναρξης και λήξης των εργασιών μας. Το τελευταίο μας βοήθησε σε μεταγενέστερη παρατήρηση της συμπεριφοράς του συστήματος για το γεγονός που μας ενδιέφερε.

Είναι σημαντικό εδώ να αναφέρουμε ότι ο σχεδιασμός του ganglia του επιτρέπει να μην προκαλεί θόρυβο στις μετρήσεις με την έννοια ότι όλα τα δομικά του στοιχεία που έχουμε αναφέρει μέχρι στιγμής, κάνουν την ελάχιστη δυνατή χρήση κύριας μνήμης καθώς και υπολογιστικού χρόνου για τις ανάγκες των εργασιών τους. Για παράδειγμα η modular σχεδίαση του gmond daemon του επιτρέπει να έχει φορτωμένα στην μνήμη μόνο τα modules εκείνα τα οποία χρειάζονται για τις μετρήσεις που έχει ορίσει ο χρήστης [19]. Ένας ακόμα παράγοντας που επιδέχεται προσοχής είναι ο ορισμός του χρονικού διαστήματος που ο gmond συλλέγει και ανακοινώνει τις μετρικές για κάθε κόμβο. Για τις εφαρμογές μας διαλέξαμε ένα διάστημα 15 δευτερολέπτων όπου αποδείχθηκε ότι αποτελεί καλή επιλογή τόσο όσο αναφορά την ακρίβεια των μετρήσεών μας, όσο και από την άποψη του ανεπιθύμητου θορύβου από το Ganglia.

Όσον αφορά την εγκατάσταση και την ενεργοποίηση του Ganglia εγκαταστήσαμε τους gmonds σε κάθε κόμβο του cluster μέσω του yum package manager που διαθέτουν οι διανομές Fedora Linux. Επίσης εγκαταστήσαμε με τον ίδιο τρόπο και τον gmetad daemon στον master του cluster καθώς επίσης και το gweb και έναν apache server μαζί με την php. Η παραμετροποίηση γίνεται τροποποιώντας configuration files που υπάρχουν για κάθε daemon. Συγκεκριμένα σε κάθε configuration αρχείο κάθε κόμβου ορίσαμε τις μετρικές που θέλουμε να ανακτώνται από το λειτουργικό σύστημα καθώς και το χρόνο που αυτό θα γίνεται. Παράλληλα περιορίσαμε την επικοινωνία στο multicast κανάλι ώστε ο εκάστοτε κόμβος να δέχεται μόνο μετρήσεις από τους κόμβους του cluster μας και όχι ενδεχομένως από κόμβους του cloud που επίσης χρησιμοποιούν το κανάλι για παρόμοιους σκοπούς και αποτελούν θόρυβο. Για τον gmetad daemon ορίσαμε τους κόμβους εκείνους τους οποίους κάνει polling για την ανάκτηση των μετρικών. Εκκινήσαμε όλους τους daemons καθώς και τον apache web server σαν υπηρεσίες του λειτουργικού από ένα τερματικό. Η παραπάνω

διαδικασία αυτούσια ακολουθήθηκε και για την εγκατάσταση και λειτουργία του Ganglia σε ένα μόνο κόμβο με τη μοναδική διαφορά ότι εκεί έχουμε την εκκίνηση μόνο ενός gmond daemon.

### 3.2.2 Apache Spark Web UI

Ένα ακόμα σημαντικό εργαλείο monitoring που χρησιμοποιήσαμε ως μέρος της διαδικασίας profiling είναι το apache spark web ui, στιγμιότυπο του οποίου φαίνεται και στο Σχήμα 3.3 παρακάτω. Το Web UI αυτό είναι μια ενσωματωμένη υπηρεσία που διαθέτει το Spark και εκκινεί μαζί με την εκτέλεση των διεργασιών master και worker σε κάθε κόμβο του cluster. Συγκεκριμένα σε κάθε κόμβο εκτελείται ένας μικρός web server που περιμένει http αιτήσεις σε συγκεκριμένες πόρτες (π.χ port 8080 για τον master). Με αυτό τον τρόπο παρέχονται monitoring μετρήσεις τόσο κατά τη διάρκεια εκτέλεσης ενός job όσο και μετά το πέρας εκτέλεσής του.


*Σχήμα 3.3: Apache Spark Web UI*

Το Spark Web UI αναλύει κάθε εφαρμογή σε jobs,stages και τελικά σε tasks σύμφωνα με αυτά που εισαγωγικά αναφέραμε για την αρχιτεκτονική εκτέλεσης του Spark. Για κάθε ένα από αυτά παρέχει πληροφορίες για την χρονική διάρκεια, την παρούσα κατάσταση καθώς και ενδεχόμενα errors και τα stack traces αυτών. Ειδικότερα για τα tasks, που είναι οι μικρότερες οντότητες υπολογισμού, μας δίνει πληροφορίες για τους συγκεκριμένους κόμβους που αυτά έχουν δρομολογηθεί ενώ ταυτόχρονα μας πληροφορεί για τη χρονική διάρκεια που απαιτήθηκε για τη δρομολόγηση τους, το deserialization χρόνο καθώς και το χρόνο ο οποίος δαπανήθηκε για την λειτουργία του garbage collector. Σε επίπεδο κόμβων μας δίνει την δυνατότητα να ελέγξουμε όλα τα log files για τις διεργασίες τόσο του master και των workers όσο και των executors. Τέλος με το Web UI μπορούμε να δούμε την

χρήση μνήμης που χρησιμοποιείται κάθε στιγμή για caching του dataset που χρησιμοποιεί η εφαρμογή μας.

Για τους σκοπούς monitoring της εργασίας μας τα παραπάνω ήταν ιδιαίτερα σημαντικά γιατί μας βοήθησαν να κατανοήσουμε την πορεία εκτέλεσης του αλγορίθμου k-means στο Spark αλλά και να διορθώσουμε τυχόν σφάλματα τόσο στο επίπεδο του κώδικα που τον υλοποιεί αλλά και στο επίπεδο των κόμβων σε περίπτωση που αντιλαμβανόμασταν μια αντικανονική συμπεριφορά.

### 3.2.3 Προγράμματα iostat και date

Επιπρόσθετα από τις μετρικές που χρησιμοποιήσαμε από το Ganglia χρειαστήκαμε εργαλεία που θα μας βοηθήσουν να παρατηρήσουμε το χρόνο εκτέλεσης κάθε εργασίας αλλά επίσης και την χρήση του δίσκου κατά τη διάρκεια αυτή. Για αυτές τις μετρήσεις προτιμήσαμε να εμπιστευθούμε τις εντολές iostat [20] και date που διαθέτουν όλες οι διανομές λειτουργικών συστημάτων Linux.

Το εργαλείο iostat είναι πρόγραμμα που εκτελείται δίνοντας την εντολή iostat σε ένα τερματικό και αναφέρει μεταξύ άλλων το μέγεθος των δεδομένων που διαβάζονται σε ένα χρονικό διαστήμα από τον τοπικό δίσκο, καθώς και το μέγεθος των δεδομένων που γράφονται. Αποτελεί κομμάτι του sysstat πακέτου που διαθέτουν όλες οι διανομές Linux μέσω των εκάστοτε software package manager με τον οποίο συνεργάζονται για την εγκατάσταση λογισμικού. Στη περίπτωση των διανομών Fedora που χρησιμοποιούμε εμείς, η εγκατάσταση του sysstat γίνεται μέσω του yum package manager. Η εντολή παρέχει μια πληθώρα επιλογών οι οποίες παραμετροποιούν το είδος και την μορφή της εξόδου. Για τις ανάγκες των δικών μας μετρήσεων η εντολή που εκτελέστηκε σε κάθε κόμβο που μας ενδιέφερε να μετρήσουμε την κίνηση στο δίσκο, ήταν η παρακάτω:

**iostat -dm 5**

Στη παραπάνω μορφή της εντολής με την επιλογή –d ζητάμε από την εντολή να αναφέρει μετρικές που σχετίζονται με την εγγραφές και αναγνώσεις στο δίσκο, ενώ με την επιλογή –m ζητάμε ο υπολογισμός να γίνει σε MB. Ο αριθμός 5 ορίζει το χρονικό βήμα με το οποίο η εντολή υπολογίζει τις παραπάνω μετρικές σε δευτερόλεπτα. Η εντολή σε αυτή τη μορφή υπολογίζει αθροιστικά σε MB των όγκο δεδομένων που διαβάζονται και εγγράφονται στο τοπικό δίσκο για το ορισμένο από το χρήστη χρονικό βήμα και το διαιρεί με αυτό για να ανακοινώσει ως αποτέλεσμα έναν μέσο όρο. Εμείς χρησιμοποιήσαμε την εντολή ανακατευθύνοντας την έξοδό της σε ένα αρχείο το οποίο ήταν ευθύνη του parser, όπως θα δούμε παρακάτω, να μορφοποιήσει κατάλληλα και να συλλέξει τα δεδομένα.

Το πρόγραμμα date διατίθεται ενσωματωμένο ως λειτουργία των τερματικών όλων των διανομών Linux. Η εντολή στη γενική της χρήση αναφέρει την εκάστοτε ημερομηνία και ώρα του συστήματος. Για τους σκοπούς της χρονομέτρησης των εργασιών μας την χρησιμοποιήσαμε στη παρακάτω μορφή:

**date +%s**

Η επιλογή "+%s" της εντολής εκτυπώνει ένα Unix timestamp σε δευτερόλεπτα. Η χρήση δύο τέτοιων timestamp στην αρχή και στο πέρας κάθε εργασίας μας, και η διαφορά μεταξύ τους μας έδινε την χρονική διάρκειά της. Η ακρίβεια της μέτρησης μας επαληθεύθηκε και από την αντίστοιχη πληροφορία που παρέχει για την κατανεμημένη εκτέλεση το Spark Web UI που είδαμε στη προηγούμενη ενότητα.

## 3.3 Συλλογή μετρικών και μορφοποίηση (parsing)

Όπως φαίνεται και από την ανάλυση που έχουμε κάνει έως τώρα από το monitoring κομμάτι έχουμε δύο πηγές μετρήσεων, αυτές που μας παρέχει το Ganglia και αυτές που παίρνουμε από τις εντολές iostat και date. Αυτές οι μετρικές έπρεπε να συγκεντρωθούν και να μορφοποιηθούν κατάλληλα ώστε να έχουμε τα τελικά δεδομένα της διαδικασίας profilling που ζητάμε.

Όσο αναφορά τη συλλογή των μετρικών ο κώδικας που την υλοποιεί επιλέξαμε να είναι αυτός που αναλάμβανει να τρέχει και τα πειράματά μας στα συστήματα που μελετούμε. Αυτό διότι η συλλογή των μετρικών του Ganglia, που όπως αναφέραμε συλλέγονται και αποθηκεύονται στην rrd, λόγω της round robin πολιτικής αποθήκευσης που ακολουθεί η ακρίβεια των μετρήσεων χάνεται μετά από το πέρας ορισμένου χρονικού διαστήματος και άρα αυτή πρέπει να γίνεται άμεσα. Ο κώδικας είναι ουσιαστικά ένα bash script το οποίο χρησιμοποιεί την διεπαφή που παρέχει η rrd database και ονομάζεται rrdtool. Το rrdtool χρησιμοποιείται και αυτό ως εντολή από κάποιο τερματικό. Για τις ανάγκες τις δικές μας χρησιμοποιήσαμε την παρακάτω μορφή της εντολής:

rrdtool fetch {metric_file} AVERAGE –start {start_time} –end {end_time}

Όπως βλέπουμε με το rrdtool μπορούμε να ανακτήσουμε τις μετρικές όπως αυτές αποθηκεύονται στα αρχεία που διατηρεί η rrd database αρκεί να δώσουμε το όνομα του αρχείου (metric_file) που μας ενδιαφέρει την χρονική στιγμή εκκίνησης των μετρήσεων καθώς και την χρονική στιγμή πέρατος των μετρήσεων (start_time και end_time). Οι προηγούμενες δύο τιμές πρέπει να δοθούν ως unix timestamps πράγμα το οποίο ταιριάζει και με τον τρόπο που επιλέξαμε να χρονομετρούμε μέσω της εντολής date.

Η συλλογή όλων των μετρικών, είτε αφορά αυτές που ανακτώνται από το Ganglia είτε αυτές που παίρνουμε από τις εντολές iostat και date, αποθηκεύονται σε αρχεία στο τοπικό δίσκο είτε του master της κατανεμημένης αρχιτεκτονικής είτε του μηχανήματος της κεντρικής υλοποίησης.

Ο parser αποτελεί ουσιαστικά μία μικρή συλλογή προγραμμάτων σε python και σε bash script που συνεργατικά αναλαμβάνει να μορφοποιήσει τα δεδομένα που πήραμε από τα προηγούμενα στάδια σε μορφή κατάλληλη για χρήση στα μοντέλα που θέλουμε να εκπαιδεύσουμε. Συγκεκριμένα οι μετρικές που λαμβάνονται από την rrd είναι στην μορφή ενός unix timestamp και μιας τιμής σε επιστημονική σημειογραφεία που δεν είναι κατάλληλη για άμεση χρήση. Για το σκοπό αυτό γράψαμε ένα script σε python που παίρνει ως είσοδο όλα τα αρχεία μετρικών για κάθε εκτέλεση πειράματος και μετατρέπει τις μετρικές μας σε αριθμούς κινητής υποδιαστολής και να τις αποθηκεύσει εκ νέου σε ένα αρχείο csv (comma separated values).

Μία ακόμα δουλειά του parser που υλοποιήθηκε με built-in εργαλεία που παρέχουν οι διανομές Linux είναι η μορφοποίηση της εξόδου της εντολής iostat σε csv αρχείο. Συγκεκριμένα με της εντολές sep και tr επιλέχθηκαν και απομονώθηκαν μόνο οι τιμές που αφορούσαν τα δεδομένα εγγραφής και ανάγνωσης από το δίσκο και αποθηκεύτηκαν και αυτά σε ένα νέο csv αρχείο.

Επίσης αναγκαία δουλειά για τον parser ήταν η εξαγωγή της μετρικής που θα ποσοτικοποιούσε τη χρήση της κύριας μνήμης κάθε στιγμή. Είδαμε σε προηγούμενη ενότητα ότι η default τιμές που παρέχει το Ganglia σχετικά με τη μνήμη είναι αυτές που αφορούν το μέγεθος αυτής που είναι ελεύθερη προς χρήση κάθε στιγμή (mem_free), το μέγεθος της μνήμης που χρησιμοποιείται ως cache (mem_cached) και το μέγεθος της μνήμης που χρησιμοποιείται ως buffers (mem_buffers). Για τον υπολογισμό της μνήμης που είναι σε χρήση κάθε στιγμή μας χρειάζεται ο παρακάτω υπολογισμός:

mem_used=mem_total - (mem_free+mem_cached+mem_buffers)

Στη παραπάνω σχέση η ποσότητα mem_total είναι σταθερή και δεδομένη, και αφορά στη διαθέσιμη συνολικά κύρια μνήμη που υπάρχει στον κόμβο για τον οποίο γίνεται η μέτρηση. Οι ποσότητες mem_cached και mem_buffers προστίθενται στην ποσότητα mem_free γιατί ο ρόλος τους είναι επικουρικός στα συστήματα Linux και θεωρούνται περιοχές της μνήμης που είναι άμεσα ανακτήσιμες από το λειτουργικό σε περίπτωση που χρειαστούν για την εκτέλεση κάποιας διεργασίας. Για τον παραπάνω υπολογισμό δημιουργήσαμε λοιπόν ένα ακόμα script σε python που αναλάμβανε να συγκεντρώσει τις μετρικές mem_free,mem_cached και mem_buffers για κάθε χρονικά στιγμή και να υπολογίσει την τιμή της μετρικής mem_used.

Μετά το πέρας όλων των παραπάνω διαδικασιών οι μετρικές μας είναι στη μορφή:

timestamp,value

για τις ανάγκες τις μοντελοποίησης όμως πρέπει για κάθε μετρική με πολλαπλές τιμές να θέσουμε μια τιμή αντιπρόσωπο. Αυτές οι μετρικές είναι εκείνες που αφορούν τις μετρήσεις IO (input/output) στο δίσκο , την χρήση της κύριας μνήμης καθώς και δικτυακής κίνησης. Για αυτό το λόγω γράψαμε ένα bash script που υπολογίζει τους μέσους όρους από τις όλες τις συλλεγμένες τιμές και θέτει αυτή την τιμή ως αντιπροσωπευτική για την εκάστοτε μέτρηση.

## 3.4 Openstack Cloud

Το περιβάλλον εγκατάστασης που χρησιμοποιήθηκε στη παρούσα εργασία είναι το σύννεφο (cloud) Openstack του εργαστηρίου cslab [21]. Ο Openstack είναι ένα cloud operating system που προσφέρει υπηρεσίες υπολογιστικών πόρων (compute), δικτύωσης (networking) και αποθήκευσης (storage) στο cloud που εγκαθίσταται.

Η υποδομή του cloud αποτελείται από 6 server μηχανήματα που χωρίζονται στις δύο κατηγορίες με τα αντίστοιχα χαρακτηριστικά, όπως αυτά φαίνονται στον Πίνακα 3.1 παρακάτω. Οι server χρησιμοποιούν τον QEMU για hypervisor που συνεργάζεται με το KVM module για να παρέχουν υπηρεσίες εικονικοποίησης υλικού (hardware virtualization) με απόδοση πολύ κοντά σε αυτή ενός φυσικού μηχανήματος. Η δημιουργία των εικονικών μηχανών γίνεται έυκολα μέσω του Openstack dashboard, που είναι ένα Web UI που παρέχει τη δυνατότητα εκκίνησης εικονικών μηχανών με επιλογή ανάμεσα σε πολλά διαφορετικά λειτουργικά συστήματα (π.χ Fedora, Ubuntu, Debian) και διαφορετικά hardware resources όπως μέγεθος δίσκου, αριθμός cores και μέγεθος κύριας μνήμης.

| Specifications | Server Type 1 (4 hosts) | Server Type 2 (2 hosts) |
|---|---|---|
| **Cpu** | Intel(R) Xeon(R) E5645 (x2)<br>6 cores – 12 hardware threads | Intel(R) Xeon(R) E5-2650 (x2)<br>8 cores – 16 hardware threads |
| **Ram** | 96 GB | 64 GB |
| **Storage** | 1TB (rootfs + vm images)<br>2x1TB (volume storage)<br>256 GB SSD | 2TB (rootfs + vm images) |
| **Network** | 2 Ethernet Interfaces (public+private) | 2 Ethernet Interfaces (public+private) |

*Πίνακας 3.2: Openstack cloud server type specifications*

Για τη παρούσα διπλωματική και για κάθε εικονική μηχανή που κατασκευάσαμε, ως κομμάτι των αρχιτεκτονικών που περιγράψαμε, χρησιμοποιήσαμε το λειτουργικό σύστημα Fedora 21, που είναι μια ελεύθερη διανομή Linux την οποία εγκαταστήσαμε σε μηχανήματα με αρχιτεκτονική συνόλου εντολών x86/64-bit. Για αποθηκευτικό μέσο επιλέξαμε volumes τα οποία είναι block device storage επιλογή που παρέχει ο Openstack, και δύναται να χρησιμοποιηθούν είτε ως boot media για το επιλεχθέν λειτουργικό σύστημα (που είναι και αυτή που χρησιμοποιήθηκε από εμάς) είτε ως ανεξάρτητο storage media που προσαρτάται σε ένα ήδη υπάρχον virtual machine.

# Κεφάλαιο 4

# Εκτέλεση και μοντελοποίηση του αλγορίθμου K-means στο Weka

Στο παρόν κεφάλαιο παρουσιάζουμε την κεντρική υλοποίηση του αλγορίθμου k-means όπως αυτή έγινε με τη χρήση της σουίτας προγραμμάτων Weka. Ξεκινάμε το κεφάλαιο περιγράφοντας συνολικά την πειραματική διαδικασία που ακολουθήσαμε για να εξάγουμε τα δεδομένα στα οποία στηριζόμαστε για τα στάδια της ανάλυσης και της μοντελοποίησης. Στη συνέχεια αφιερώνουμε μία ενότητα στη περιγραφή των datasets που χρησιμοποίησαμε καθώς και τον τρόπο κατασκευής τους. Αφιερώνουμε επίσης μια ενότητα που περιγράφουμε τον τρόπο με τον οποίο εκτελείται ο αλγόριθμος στο Weka ενώ στις τελευταίες δυο ενότητες του κεφαλαίου αναλύουμε και παρουσιάζουμε τα αποτελέσματα των πειραμάτων μας. Τέλος παραθέτουμε τα μοντέλα που κατασκευάσαμε και επαληθεύουμε την ακρίβειά τους.

## 4.1 Περιγραφή πειραματικής διαδικασίας

Σκοπός της πειραματικής διαδικασίας είναι να εκτελέσουμε τον αλγόριθμο k-means διαφοροποιώντας σε κάθε εκτέλεση παραμέτρους που σχετίζονται με τη λογική του αλγορίθμου, όπως αυτές είναι η επιλογή του k ως αριθμό clusters, ο αριθμός των επαναλήψεων του αλγορίθμου μέχρι αυτός να επιτύχει σύγκλιση, καθώς και παραμέτρων που περιγράφουν τη μορφή του dataset εισόδου όπως αυτές είναι ο αριθμός των διανυσμάτων/αντικειμένων (instances) και των διαστάσεων (dimensions) του κάθε instance του dataset. Παράλληλα ενδιαφέρον παρουσιάζει και η εναλλαγή του περιβάλλοντος εκτέλεσης από την άποψη του κύριας μνήμης του μηχανήματος που εκτελείται ο αλγόριθμος αλλά και του αριθμού των πυρήνων (cores) που διαθέτει. Για κάθε τέτοια εκτέλεση μας ενδιαφέρει η συλλογή μετρικών απόδοσης όπως ήδη έχουμε αναφέρει, που θα μας βοηθήσει σε πρώτη φάση να αναλύσουμε την συμπεριφορά που παρατηρούμε και παράλληλα να την μοντελοποιήσουμε.

Από μαθηματική σκοπιά η παραπάνω διαδικασία μας φέρνει αντιμέτωπους με έναν χώρο πειραμάτων που αποτελεί το καρτεσιανό γινόμενο των πιθανών τιμών που μπορούν να

λαμβάνουν οι προαναφερθείσες παράμετροι. Για παράδειγμα μας ενδιαφέρει να γνωρίζουμε την εκτέλεση ενός dataset με 10000 instances που έχουν 50 dimensions το καθένα με 10 clusters και 100 επαναλήψεις (iterations) σε ένα μηχάνημα με 4GB ram και 2 cores. Παράλληλα θέλουμε να παρατηρήσουμε και να συγκρίνουμε την συμπεριφορά εκτέλεσης σε περίπτωση που μεταβάλλουμε μία παράμετρο από τις παραπάνω ή και περισσότερες. Όπως γίνεται εύκολα κατανοητό ο πειραματικός χώρος που προσπαθούμε να σκιαγραφήσουμε εκτείνεται πολύ γρήγορα σε μεγέθη που δεν είναι εύκολα παρατηρήσιμα και διαχειρίσιμα.

Για τον παραπάνω λόγο βαρύνουσας σημασίας στη διαδικασία της μελέτης μας, είναι η προσεκτική επιλογή εκείνων των δειγμάτων (samples) του παραπάνω χώρου[22] (π.χ. adaptive και uniform sampling ), που θα μας προσφέρουν την πιο αντιπροσωπευτική εικόνα για την εκτέλεση του αλγορίθμου και παράλληλα θα μας αποτρέψουν να κάνουμε πυκνή δειγματοληψία που θα στερείται χρήσιμης πληροφορίας και νοήματος μελέτης. Ένας ακόμα παράγοντας που μας βοηθάει προς αυτή τη κατεύθυνση είναι και η παρατήρηση παραμέτρων που δεν έχουν καμία επίδραση στις μετρικές που μελετούμε και άρα η εύλογη επιλογή μη χρησιμοποίησης διαφορετικών τιμών τους στη πειραματική διαδικασίας.

Έχοντας τα παραπάνω υπόψη οι επιλογές μας όσο αναφορά τα διαφορετικά μηχανήματα εκτέλεσης ήταν αυτές που φαίνονται στο Πίνακα 4.1 που παραθέτουμε εδώ:

| Flavor Types | Main Memory (GB) | Number of Cores |
|---|---|---|
| Type 1 | 4 | 4 |
| Type 2 | 8 | 4 |
| Type 3 | 16 | 8 |

*Πίνακας 4.1: Τύποι μηχανημάτων υλοποίησης Weka*

Οι επιλογές των παραπάνω τύπων μηχανημάτων έγιναν με δύο κριτήρια. Αρχικά περιοριστήκαμε να χρησιμοποιήσουμε δύο μόνο διαφορετικές εκδοχές για το συνολικό αριθμό cores των μηχανημάτων γιατί όπως θα αποδείξουμε και σε επόμενη ενότητα η εκτέλεση του αλγορίθμου είναι ανεξάρτητη από τα cores. Όσο αναφορά την επιλογή της μνήμης αυτή έγινε μετά από τους αρχικούς μας πειραματισμούς όπου παρατηρήσαμε ότι εκτελέσεις του αλγορίθμου με τιμές μνήμης μεγαλύτερης των 16 GB που θέσαμε ως μέγιστη δεν είχαν πρακτική αξία μελέτης καθώς υπερβαίνουν τόσο τα λογικά πλαίσια χρήσης μνήμης μιας κεντρικής υλοποίησης αλλά και τα ανεκτά χρονικά περιθώρια που ανταποκρίνονται σε ρεαλιστικές εφαρμογές.

Μεγαλύτερο ίσως ενδιαφέρον είχε η προσεκτική επιλογή των διαφορετικών datasets που χρησιμοποιήσαμε για όπως αυτές περιγράφονται από τον αριθμό των instances αλλά και των dimensions. Όπως είναι φυσικό αυτές οι δύο παράμετροι ορίζουν το συνολικό μέγεθος του dataset και ταυτόχρονη αύξηση αυτών μπορεί να οδηγήσει σε μεγέθη που η υλοποίηση του Weka δεν μπορεί να διαχειριστεί αποδοτικά και η μόνη χρήσιμη πληροφορία που μπορούν να μας προσφέρουν είναι να ορίσουμε ένα ανώτατο αποδεκτό όριο μεγέθους dataset. Παράλληλα όμως αυτές οι δύο παράμετροι είναι από τις πιο ενδιαφέρουσες όπως θα δούμε, και άρα έπρεπε να γίνει ένας συμβιβασμός ανάμεσα στις επιλογές του εύρους τιμών των δύο αυτών παραμέτρων. Στη πειραματική διαδικασία που ακολουθήσαμε αποφασίσαμε

να επιλέξουμε λίγο μεγαλύτερο πλήθος τιμών για τα dimensions. Όσο αναφορά στην επιλογή των διαφορετικών cluster ανά εκτέλεση του αλγορίθμου εκεί επιλέξαμε να έχουμε ένα μεγαλύτερο εύρος τιμών με πυκνότερη δειγματοληψία ανάμεσα σε αυτό το εύρος. Η επιλογή μας αυτή σχετικά με τα clusters βασίστηκε στη αρχική μας παρατήρηση ότι η μεταβολή αυτής της παραμέτρου έχει σημαντικές επιπτώσεις στη χρονική εκτέλεση καθώς και στο γεγονός ότι γενικότερα στις πραγματικές υλοποιήσεις του αλγορίθμου υπάρχει μεγάλη διαφοροποίηση στις τιμές των clusters. Όλα τα παραπάνω φαίνονται και στον Πίνακα 4.2 που παραθέτουμε όλες τις διαφορετικές τιμές που επιλέξαμε για αριθμό instances,dimensions και clusters.

| Τιμές πλήθους instances | Τιμές πλήθους dimensions | Τιμές Clusters (K) |
|---|---|---|
| 10000,2500,50000,75000,100000 | 10,50,100,250,500,750,1000 | 10,100,200,…,1600 |

*Πίνακας 4.2: Τιμές instances,dimensions και clusters υλοποίησης Weka*

Για τις επιλογές των τιμών προσπαθήσαμε κατά το δυνατό να πραγματοποιήσουμε ομοιόμορφη δειγματοληψία ανάμεσα στην ελάχιστη και μέγιστη τιμή. Η παραπάνω επιλογή έγινε στη βάση του ότι η ομοιόμορφη δειγματοληψία μας αποτελεί και μια διαισθητικά σωστή επιλογή αλλά και εύκολη. Σκοπός μας ήταν να ενισχύσουμε την δειγματοληψία σε σημεία του πειραματικού χώρου που μπορεί να παρατηρούσαμε μια πιο ενδιαφέρουσα συμπεριφορά εκτέλεσης μετά το πέρας και της ανάλυσης των εξαγόμενων αποτελεσμάτων. Απεδείχθει τελικά ότι η επιλογή της ομοιόμορφης δειγματοληψίας επαρκούσε για την ανάλυση και μοντελοποίηση που θέλαμε να κάνουμε.

Τέλος ένα ακόμα σημαντικό στοιχείο που έπρεπε να λάβουμε υπόψη για την επιλογή των διαφορετικών πειραμάτων ήταν το συνολικό πλήθος των αποτελεσμάτων που θα λάβουμε. Το τελευταίο έχει μεγάλη αξία για την μοντελοποίηση που θέλουμε να επιτύχουμε. Αυτό διότι η ακρίβεια και η αξία ενός μοντέλου που περιγράφει μια διαδικασία έχει στενή εξάρτηση τόσο με το μέγεθος του training set με το οποίο το εκπαιδεύουμε αλλά και με την ύπαρξη ενός test set με το οποίο να μπορεί να επαληθευθεί η προβλεπτική του αξία. Όπως είναι φυσικό ο διαχωρισμός σε αυτά τα δύο είδη δεδομένων ευνοείται όταν διαθέτουμε ένα αρκούντως μεγάλο μέγεθος δεδομένων επίδοσης ως αποτέλεσμα των πειραμάτων μας. Η παραπάνω επιλογή τιμών ικανοποιεί αυτή την απαίτηση αφού συνολικά έχουμε:

**<u>Συνολικές Εκτελέσεις:</u> 5 (# of Instances) x 7 (# of dimensions) x 14 (# of clusters) = 490 εκτελέσεις**

## 4.2 Κατασκευή των datasets

Όπως είχαμε αναφέρει και εισαγωγικά σχετικά με τα δεδομένα εισόδου του αλγορίθμου k-means, σε πραγματικές εφαρμογές αποτελούν αντικείμενα των οποίων τα χαρακτηριστικά απεικονίζονται σε διαφορετικές διαστάσεις του κάθε instance και ως τιμές έχουν αριθμούς που αναλαμβάνουν να ποσοτικοποιήσουν το εκάστοτε χαρακτηριστικό. Με

βάση αυτά και δεδομένου της απουσίας πραγματικών δεδομένων που να μπορούμε να χρησιμοποιήσουμε για συσταδοποίηση γράψαμε ένα python script το οποίο αναλαμβάνει την κατασκευή dataset με συγκεκριμένο αριθμό από instances αλλά και dimensions που ορίζονται από το χρήστη ως παράμετροι εισόδου στο script.

Μερικά ακόμα χαρακτηριστικά του dataset που τροφοδοτείται στον αλγόριθμο είναι το εύρος των επιτρεπόμενων τιμών το οποίο χαρακτηρίζει το κάθε χαρακτηριστικό-διάσταση καθώς και το είδος της τιμής (πχ ακέραιος, κινητής υποδιαστολής). Τα παραπάνω είναι παράγοντες που επηρεάζουν ευθέως την απόδοση του αλγορίθμου γιατί αν για παράδειγμα έχουμε ένα instance που αποτελείται από αριθμούς κινητής υποδιαστολής, τότε στο υπολογιστικό βήμα του αλγορίθμου που υπολογίζει την απόσταση του instance από κάθε κεντροειδές μπορεί να εισέρχεται μικρή χρονική επιβάρυνση που πολλαπλασιασμένη στο μέγεθος των instances και των dimensions μπορεί να επηρεάζει επι παραδείγματι τη συνολική χρονική επίδοση.

Ακόμα ίσως πιο σημαντικός παράγοντας που έπρεπε να λάβουμε υπόψη μας ήταν το πως θα επιλέξουμε να κατανέμονται τα παραπάνω διανύσματα-αντικείμενα στον N-διάστατο ευκλείδιο χώρο. Αυτό γιατί η σύγκλιση του αλγορίθμου για δεδομένη κατανομή μπορεί, σε συνδυασμό πάντα με την κατάλληλη αρχική τιμή των κεντροειδών, να οδηγήσει σε πολύ γρήγορη ή αντίστοιχα πολύ αργή σύγκλιση. Για παράδειγμα ένα dataset που έχει κατανομή διανυσμάτων στο χώρο που εκ φύσης ορίζουν κάποια cluster είναι λογικό σε αυτή τη περίπτωση να έχουμε πολύ πιο γρήγορη σύγκλιση. Ο αλγόριθμος K-means με τον τρόπο που διατίθεται από τη βιβλιοθήκη του Weka δίνει τη δυνατότητα ορισμού μέγιστου αριθμού επαναλήψεων, εν αντιθέσει ίσως μιας επιλογής ακριβούς αριθμού επαναλήψεων, και άρα στα αποτελέσματα των πειραμάτων μας θα επεισερχόταν ένας ακόμα παράγοντας πολυπλοκότητας.

Τα παραπάνω επιλέξαμε να τα διευθετήσουμε φτιάχνοντας, με το προαναφερθέν python script, datasets που αποτελούνταν από instances με ακέραιες τιμές που παράγονταν ως ψευδοτυχαίοι αριθμοί με εύρος από το μικρότερο ακέραιο αριθμό που μπορεί να παράγει μία 64-bit αρχιτεκτονική μέχρι το μεγαλύτερο αντίστοιχο αριθμό. Αυτοί οι δύο αριθμοί παρέχονται από το module sys της python με την κλήση sys.maxint. Οι ψευδοτυχαίοι αριθμοί παρήχθησαν με την βοήθεια του module random που παρέχει την μέθοδο randint για την παραγωγή ψευδοτυχαίων ακεραίων. Η μορφή της εντολής που παράγει τους αριθμούς αυτούς φαίνεται παρακάτω:

number= random.randint(-sys.maxint,sys.maxint)

Με τον παραπάνω τρόπο αρχικά επιλέξαμε να μελετήσουμε αποκλειστικά την συμπεριφορά του αλγορίθμου μόνο για ακεραίους. Επίσης με την παραγωγή ψευδοτυχαίων τιμών διαστάσεων με το παραπάνω εύρος τιμών δημιουργήσαμε datasets τα οποία δεν έχουν καμία 'προδιάθεση' συσταδοποίησης ούτε κάποια συγκεκριμένη μορφή στο χώρο. Με αυτό τον τρόπο κατασκευής εγγυόμαστε, όπως αποδείχθηκε και στη πράξη, ότι ο αλγόριθμος δεν θα συγκλίνει ποτέ ικανοποιητικά ώστε να τερματίσει πριν τον μέγιστο αριθμό επαναλήψεων. Η επιλογή αυτή έγινε συνειδητά ώστε οι μετρήσεις μας (κυρίως όσο αναφορά μετρούμενη χρονική επίδοση της κάθε εκτέλεσης) να απαλλαχθούν από τον παράγοντα τυχαιότητας που θα ενυπήρχε αν γινόταν καλή ή κακή αρχική επιλογή κεντροειδών που να ευνοεί ταχύτερα ή βραδύτερα αντίστοιχα, αποτελέσματα.

Οι παρατηρήσεις που αναφέρουμε παραπάνω επαληθεύθηκαν με την κατασκευή dataset όπου γινόταν τεχνητή κατασκευή προκαθορισμένων συστάδων που επιλέγονταν από την αρχή και παράγονταν ισόποσα instances γύρω από το κάθε κεντροειδές σύμφωνα σε τυχαία οριζόμενη απόσταση που ακολουθούσε κανονική κατανομή Gauss. Σε αυτά τα datasets αποδείχθηκε πειραματικά ότι υπήρχαν σημαντικές διαφορές χρονικής επίδοσης ανάμεσα σε εκτελέσεις με τις ίδιες παραμέτρους και για αυτό η επιλογή αυτή απορρίφθηκε οριστικά ως ενδεχόμενο.

## 4.3 Αλγόριθμος K-means στο Weka

Η υλοποίηση του αλγορίθμου K-means στο Weka είναι γενικά μια απλή διαδικασία. Η συγγραφή του προγράμματος γίνεται στη γλώσσα Java στην οποία το Weka έχει γραφτεί και διαθέτει και την προγραμματιστική του διεπαφή.

Ο κώδικας του χρήστη ορίζει ένα αρχείο το οποίο περιέχει το dataset προς συσταδοποίηση. Στη περίπτωσή μας αυτό ήταν ένα csv αρχείο όπως προείπαμε και με την βοήθεια της κλάσης CSVLoader που διαθέτει το Weka διαβάζουμε τα δεδομένα. Κάθε instance του dataset αντιπροσωπεύεται από την συνονόματη κλάση Instance του Weka που παρέχει μεθόδους για την αποδοτική διαχείριση των τιμών που αποθηκεύονται σε ένα instance.

Ο κώδικας του χρήστη κάνει χρήση του αντικειμένου SimpleKMeans όπου παρέχει μεθόδους με τις οποίες ορίζει τις επιθυμητές παραμέτρους του αλγορίθμου. Συγκεκριμένα με την μέθοδο setNumClusters ορίζεται ο ζητούμενος αριθμός των clusters που ζητάμε ενώ με τη μέθοδο setMaxIterations ορίζεται ο μέγιστος αριθμός επαναλήψεων που εκτελεί ο αλγόριθμος και μετέπειτα τερματίζει. Με τη μέθοδο buildClusterer στην οποία παρέχουμε το dataset όπως αυτό αποτελείται από σύνολο αντικειμένων Instances, εκκινεί η εκτέλεση του αλγορίθμου.

Στην αρχή ο αλγόριθμος δημιουργεί ένα hash table στο οποίο αποθηκεύει τα αρχικά κεντροειδή που επιλέγει με τυχαίο τρόπο από το σύνολο των instances αποφεύγοντας με αυτό τον τρόπο να επιλέξει 2 φορές το ίδιο και μετέπειτα ακολουθούνται τα δύο επαναληπτικά βήματα του αλγορίθμου όπως περιγράψαμε στο εισαγωγικό κεφάλαιο. Τέλος με τις μεθόδους getClusterCentroids και getClusterSizes μπορούμε να ανακτήσουμε τα τελικά κεντροειδή και καθώς και το πλήθος των instances που αντιστοιχίζονται στο cluster που αυτά ορίζουν.

## 4.4 Πειραματικά αποτελέσματα

Σε αυτή την υποενότητα παρουσιάζουμε και αναλύουμε τα αποτελέσματα των εκτελέσεων του K-means για τις διάφορες παραμέτρους που είδαμε στην ενότητα 4.1. Επικεντρωνόμαστε στην παρατήρηση της χρήσης cpu κατά τη διάρκεια εκτέλεσης, της εγγραφής και αναγνωσης δεδομένων στο δίσκο, της χρονικής εκτέλεσης καθώς και την χρησιμοποιούμενης κύριας μνήμης και των μεταβολών των παραπάνω μετρικών καθώς εναλλάσουμε τις παραμέτρους εκτέλεσης.

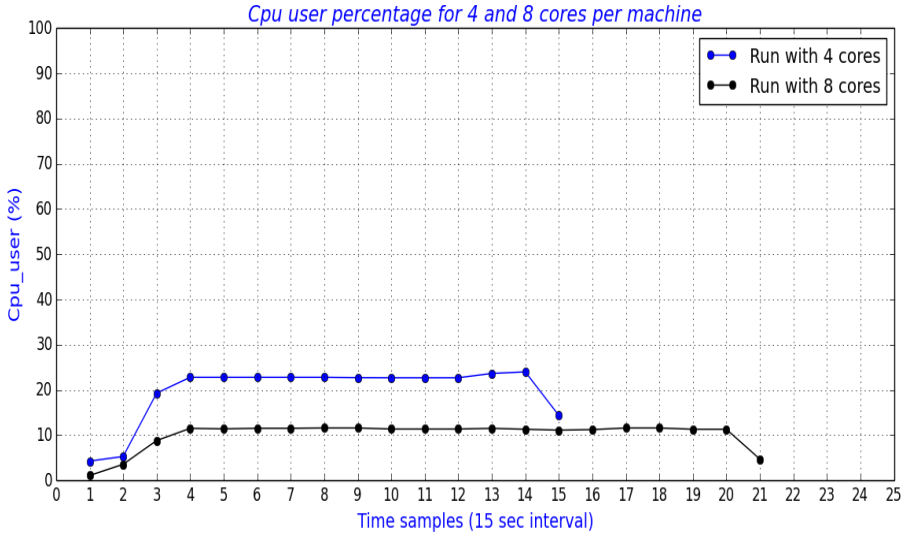
### 4.4.1 Εξάρτηση από αριθμό επαναλήψεων

Ο αριθμός των επαναλήψεων της εκάστοτε εκτέλεσης όπως είναι αναμενόμενο προτού καν εμβαθύνουμε σε περαιτέρω ανάλυση, επηρεάζει αποφασιστικά τον μετρούμενο χρόνο εκτέλεσης ο οποίος είναι και μία από τις σημαντικές μετρήσεις για τις οποίες ενδιαφερόμαστε στην μελέτη μας. Παρόλα αυτά και με γνώμονα αυτά που αναφέραμε στην παράγραφο 4.1 σχετικά με την έκταση του πειραματικού χώρου και τη δυσκολία μελέτης του, κρίναμε σκόπιμο να περιορίσουμε τη μελέτη μας σε εκτελέσεις του αλγορίθμου που αφορούν συγκεκριμένο αριθμό μέγιστων επαναλήψεων επιλέγοντας για κάθε εκτέλεση 10 επαναλήψεις. Αυτό βεβαίως δεν αποτελεί αυθαίρετη επιλογή που μας διευκόλυνε, αλλά προκύπτει από την παρατήρησή μας ότι το χρονικό κόστος κάθε επανάληψης του αλγορίθμου μπορεί (και επαληθεύθηκε πειραματικά) να θεωρηθεί σταθερό. Με γνώμονα αυτό, είναι φυσικό να αναμένουμε ότι η χρονική επίδοση του αλγορίθμου για περισσότερες επαναλήψεις από τις 10 που ορίσαμε εμείς, επηρεάζεται κατά ένα μόνο σταθερό παράγοντα που εξαρτάται από το κατά πόσες φορές μεγαλύτερες είναι οι επαναλήψεις σε σχέση με τις μετρούμενες 10. Ταυτόχρονα επαληθεύθηκε και η διαισθητική μας αντίληψη ότι οι υπόλοιπες μετρικές ενδιαφέροντος που αναφέραμε είναι πλήρως ανεξάρτητες τις επιλογής των επαναλήψεων του αλγορίθμου. Για το υπόλοιπο λοιπόν της μελέτης θεωρούμε δεδομένο ότι κάθε εκτέλεση αφορούσε 10 επαναλήψεις.

### 4.4.2 Μέτρηση CPU

Κατά τη διάρκεια όλων των εκτελέσεων που επιχειρήσαμε μετρήσαμε την χρήση της cpu μέσω των μετρικών cpu_wio, cpu_user και cpu_system που είδαμε στην ενότητα 3.2.1. Όσο αναφορά το cpu_wio δεν παρατηρήθηκαν αξιοσημείωτες εναλλαγές των τιμών του που να υποδηλώνουν ότι κατά τη διάρκεια εκτέλεσης η υλοποίηση υποφέρει από χρόνο που καταναλώνεται ως αναμονή για εξυπηρέτηση μεταφοράς δεδομένων από και προς το δίσκο. Το παραπάνω παρατηρήθηκε ανεξαιρέτως παραμέτρων υλοποίησης αλλά και μηχανήματος. Μικρή μόνο αύξηση του ποσοστού cpu_wio παρατηρείται κατά την εκκίνηση του αλγορίθμου που όπως εξηγήσαμε διαβάζει από το δίσκο το dataset εισόδου. Σε αντίστοιχες παρατηρήσεις καταλήξαμε και για την μέτρηση cpu_system, όπου και πάλι είδαμε αμελητέα μικρές τιμές κατά τη διάρκεια εκτέλεσης.

Αντιθέτως όμως η μέτρηση cpu_user μας προσέφερε χρήσιμη πληροφορία όσο αναφορά το βαθμό παραλληλισμού της εκτέλεσης. Συγκεκριμένα παρατηρήσαμε για εκτελέσεις του αλγορίθμου σε διαφορετικά μηχανήματα με διαφορετικά cores ότι η υλοποίηση χρησιμοποιεί μόνο ένα από τα διαθέσιμα cores του μηχανήματος. Η παραπάνω παρατήρηση φαίνεται και στο Σχήμα 4.1 παρακάτω:




*Σχήμα 4.1: Ποσοστό cpu user για διαφορετικά cores/machine*

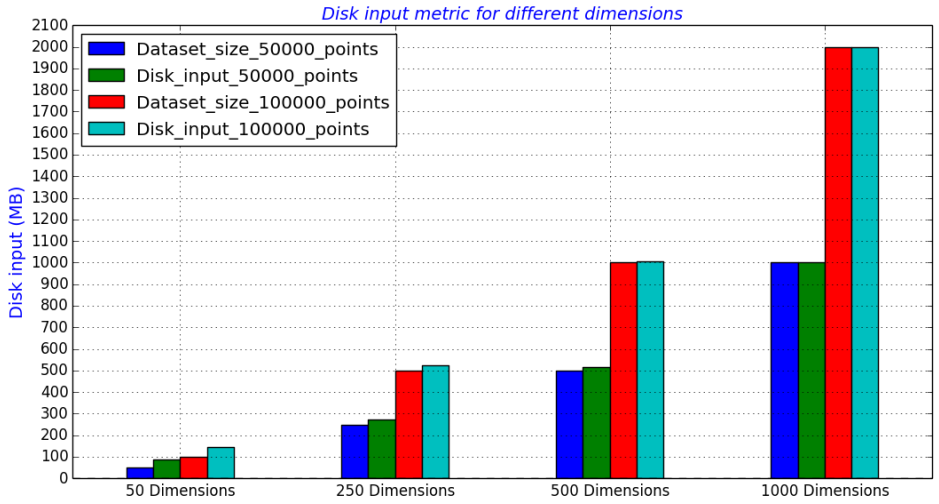
Στο παραπάνω γράφημα βλέπουμε τα ποσοστά cpu_user συναρτήσει των χρονικών δειγμάτων που πήραμε και καταγράφηκαν για το dataset με 50000 instances και 10 dimensions σε μηχάνημα με 4 cores και του dataset με 75000 instances και 10 dimensions σε μηχάνημα με 8 cores (διαφορετική διάρκεια εκτέλεσης για το καθένα). Η μέτρηση cpu_user εκφράζει σε ποσοστό χρόνου την χρήση όλων συνολικά των cores, αυτό σημαίνει ότι σε μηχάνημα με 4 cores μία μέτρηση 25% σημαίνει ότι το ένα core εκτελεί κώδικα χρήστη σε αποκλειστικό βαθμό, αντίστοιχα για ένα μηχάνημα με 8 cores ένα ποσοστό 12%-13% σημαίνει ότι ένα core είναι επίσης αφιερωμένο στην εκτέλεση κώδικα user space. Με βάση αυτά παρατηρούμε ότι όντως για την εκτέλεση με τα 4 cores παραπάνω έχουμε ένα ποσοστό που αγγίζει το 25% ενώ για το μηχάνημα με τα 8 cores έχουμε ένα ποσοστό που είναι περίπου 12%, και οι δύο μετρήσεις αφορούν το κυρίως σώμα εκτέλεσης του αλγορίθμου.

Τα παραπάνω έχουν διττή σημασία για τη περαιτέρω μελέτη μας. Αρχικά μας υποδεικνύουν ότι ο αλγόριθμος στο Weka δεν είναι παραλληλοποίησιμος και άρα η συμπεριφορά εκτέλεσης δεν εναλλάσεται με την αλλαγή των cores/μηχάνημα. Επίσης το γεγονός αυτό μας υποδεικνύει ότι υπάρχει και ανεξαρτησία της χρονικής επίδοσης από το περιβάλλον εκτέλεσης.

### 4.4.3 Disk input/output

Το disk input/output μετρήθηκε ως το σύνολο των MB που αναγνώστηκαν (input) και γράφτηκαν (output) στο δίσκο κατά τη διάρκεια εκτέλεσης. Για το disk output παρατηρήθηκε ότι ανεξαρτήτως των εκτελέσεων, υπάρχει μια μικρή αμελητέα ποσότητα MB που γράφονται στο δίσκο η οποία όμως δεν σχετίζεται τόσο με την εκτέλεση του αλγορίθμου. Αφορά κυρίως κίνηση στο δίσκο που ενυπάρχει ούτως ή αλλιώς και σχετίζεται με διαδικασίες και ανάγκες του λειτουργικού συστήματος.

Για το diks input που μετρήθηκε παραθέτουμε τα αποτελέσματα στο παρακάτω Σχήμα 4.2 για δύο διαφορετικά dataset σε σχέση με τον αριθμό των instances, και για τέσσερις διαφορετικές διαστάσεις. Το μέγεθος των cluster δεν σχετίζεται με το disk input οπότε μπορεί να θεωρηθεί ότι στο παρακάτω γράφημα αυτή η παράμετρος αντιπροσωπεύει οποιοδήποτε αριθμό cluster από αυτούς που έχουμε διαλέξει για τα πειράματά μας.




*Σχήμα 4.2: Disk input για διαφορετικό σύνολο instances και διαφορετικά dimensions*

Σημειώνουμε εδώ ότι οι disk_input μπάρες αφορούν στις μετρήσεις μας ενώ δίπλα τους ακριβώς έχουμε παραθέσει γραφικά και το μέγεθος σε MB του αντίστοιχου dataset που αποτελεί είσοδο στην συγκεκριμένη εκτέλεση. Όπως βλέπουμε το disk input αντιστοιχεί σχεδόν πλήρως σε μέγεθος με το αντίστοιχο μέγεθος του dataset ανεξαρτήτως του πλήθους των instances και των dimensions που το χαρακτηρίζουν. Αμελητέες αποκλίσεις παρατηρούνται στα μικρότερα dimensions όπου η μέτρηση είναι ελαφρώς μεγαλύτερη και προκύπτει από το γεγονός ότι το Weka χρειάζεται να φορτώσει τις απαραίτητες βιβλιοθήκες για την εκτέλεση. Όσο το μέγεθος του dataset γίνεται μεγαλύτερο η διαφορά αυτή γίνεται μη συγκρίσιμη με το dataset και παρατηρούμε πλήρη ισορροπία με την μετρούμενη τιμή. Από τα παραπάνω προκύπτει ότι μπορούμε με μεγάλη βεβαιότητα να αναμένουμε disk input μετρήσεις πάντα ισόποσες με το μέγεθος του dataset που τροφοδοτούμε στο Weka.

### 4.4.4 Χρόνος Εκτέλεσης

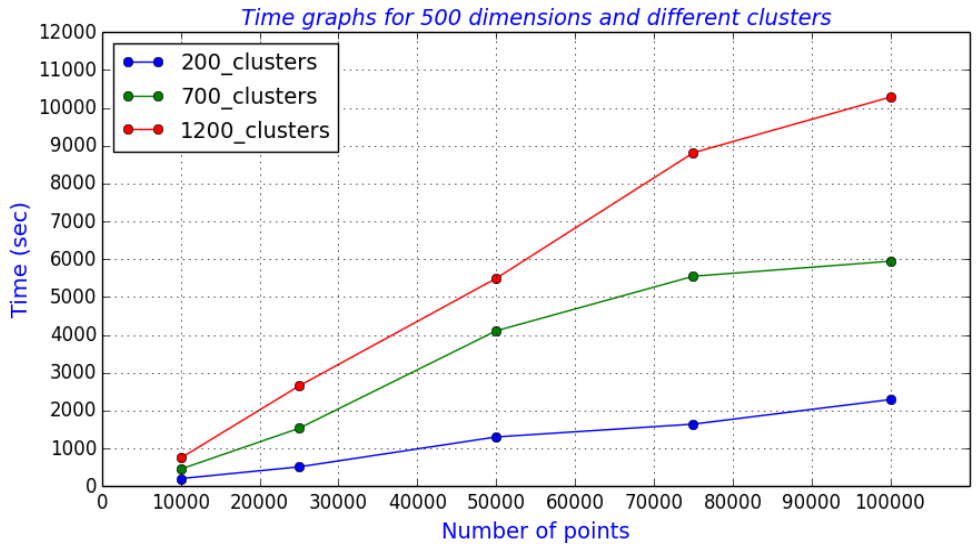
Σε αυτή την υποενότητα παρουσιάζουμε τις μετρήσεις που κάναμε σχετικά με την χρονική διάρκεια εκτέλεσης του κάθε πειράματος της κεντρικής υλοποίησης. Μακροσκοπικά η μέτρηση του χρόνου εκτέλεσης παρατηρήσαμε ότι εξαρτάται τόσο από το αριθμό των instances του dataset αλλά και από τα dimensions και τα cluster που ζητάμε από κάθε εκτέλεση. Για αυτό το λόγο η παρουσίαση των αποτελεσμάτων και η επιμέρους ανάλυση θα γίνει σε 3 στάδια-υποενότητες που αφορούν το καθένα την μεταβολή μιας από τις τρεις προαναφερθείσες παραμέτρους και τις αντίστοιχες παρατηρήσεις για τον χρόνο.

Κρίνουμε επίσης σκόπιμο να παραθέσουμε εισαγωγικά εδώ τα ερωτήματα που θα προσπαθήσουμε να απαντήσουμε στις επόμενες 3 ενότητες και αποτελούν οδηγό των ενδιάμεσων παρατηρήσεών μας αλλά και των τελικών μας συμπερασμάτων. Αυτά είναι:

- Πως και με ποιό τρόπο επηρεάζεται ο χρόνος εκτέλεσης από τη μεταβολή του αριθμού των instances του dataset , του αριθμού των dimensions και του αριθμού των clusters κάθε εκτέλεσης ξεχωριστά;
- Μπορούμε να καταλήξουμε σε κάποιο συμπέρασμα σχετικά με το ποιά από τις παραπάνω παραμέτρους έχει τον μεγαλύτερο αντίκτυπο;
- Παρατηρούμε κάποια τιμή του χρόνου εκτέλεσης που αποτελεί ξεχωριστή ειδική περίπτωση (outlier);

### 4.4.4.1 Επίδραση πλήθους instances

Για να μπορέσουμε να αναλύσουμε τις διαφορετικές τιμές που παρατηρούμε στο χρόνο εκτέλεσης σε σχέση με τη μεταβολή του αριθμού του πλήθους των instances παραθέτουμε στο Σχήμα 4.3 γραφικά 3 διαφορετικές περιπτώσεις που αφορούν συγκεκριμένη επιλογή στον αριθμό των dimensions (500) και 3 διαφορετικές τιμές του αριθμού των clusters (200,700,1200) της κάθε εκτέλεσης.



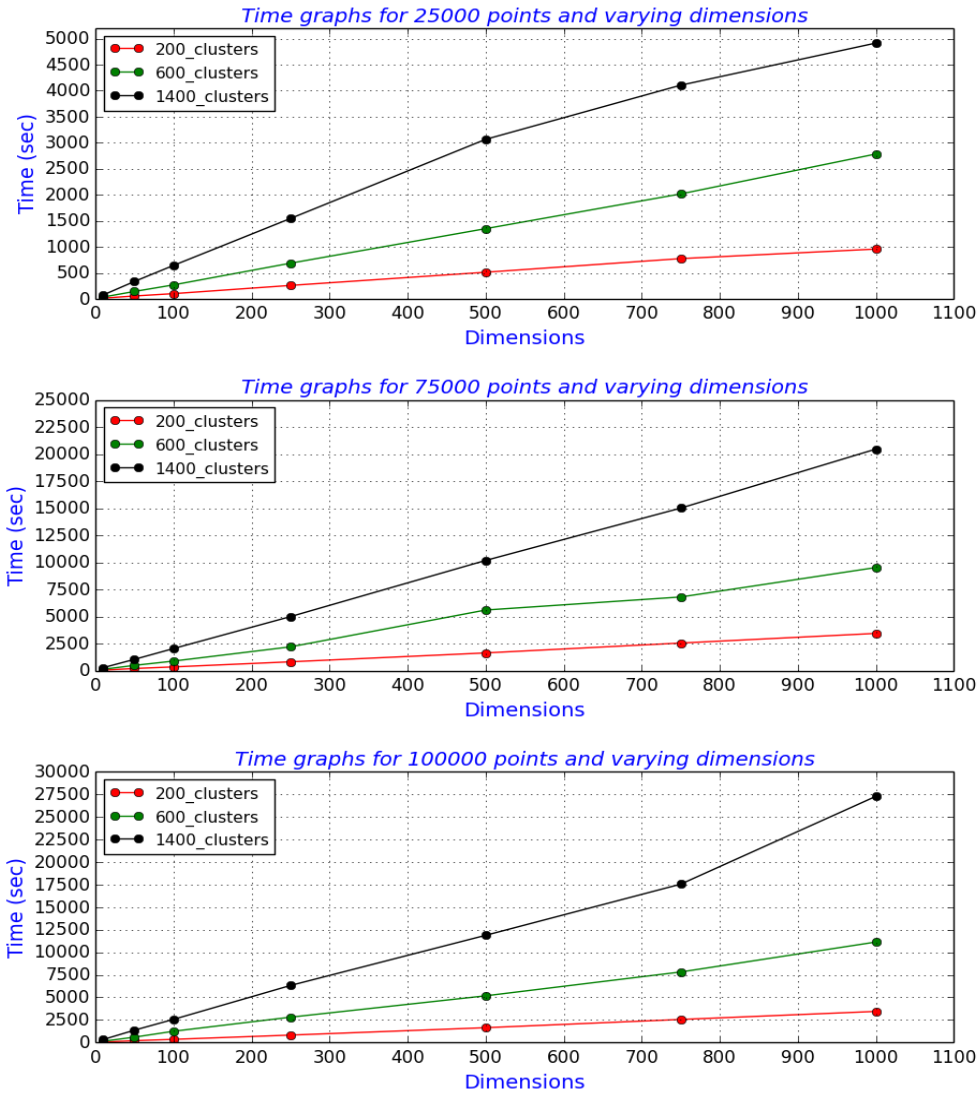

*Σχήμα 4.3: Επίδραση χρόνου εκτέλεσης για διαφορετικό πλήθος instances*

Όπως φαίνεται και από το σχήμα η εξάρτηση του χρόνου είναι γνησίως αύξουσα σε σχέση με τον αριθμό των instances και μάλιστα πλήρως γραμμική. Η γραμμικότητα αυτή είναι ανεξάρτητη του αριθμού των cluster κάθε εκτέλεσης όπως βλέπουμε από τις τρεις διαφορετικές ευθείες που αντιπροσωπεύουν διαφορετικές τιμές του. Παρόμοια ανεξαρτησία παρατηρούμε και για διαφορετικές τιμές του πλήθους των dimensions αλλά για λόγους χώρου επιλέξαμε να δείξουμε μονάχα την περίπτωση των 500 dimensions. Μια ελάχιστα μικρή απόκλιση από τη γενικότερη γραμμική συμπεριφορά ίσως μπορούμε να

παρατηρήσουμε για την περίπτωση των 700 clusters που όμως μπορεί να αποδοθεί εν μέρει και στη στοχαστικότητα του περιβάλλοντος εκτέλεσης.

#### 4.4.4.2 Επίδραση πλήθους διαστάσεων

Για την επίδραση του πλήθους των διαστάσεων επικεντρωνόμαστε στη μελέτη 3 dataset με 25000,75000 και 100000 instances και 3 διαφορετικά cluster 200,600 και 1400. Τα αποτελέσματα απεικονίζονται γραφικα στο Σχήμα 4.4 παρακάτω:

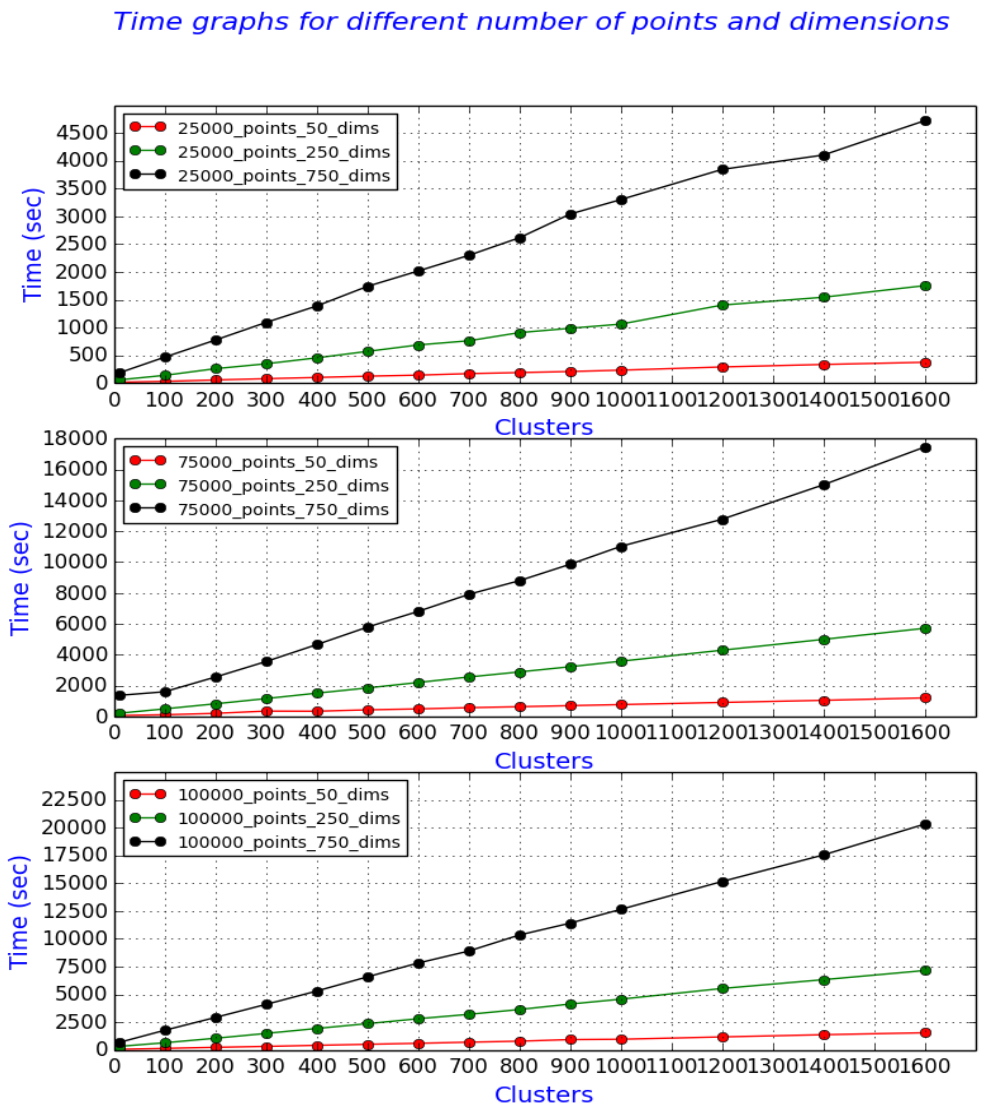



*Σχήμα 4.4: Επίδραση χρόνου εκτέλεσης για διαφορετικό πλήθος dimensions*

Στη περίπτωση των dimensions παρατηρούμε εκ νέου γνησίως αύξουσα και γραμμική συμπεριφορά του χρόνου εκτέλεσης. Αυτή η εικόνα διατηρείται ανεπηρέαστη τόσο από την εναλλαγή της τιμής του πλήθου των instances αλλά και του πλήθους των clusters σε κάθε εκτέλεση. Κάτι που επίσης παρατηρούμε, είναι ότι καθώς αυξάνονται οι τιμές του πλήθους

των instances το εύρος τιμών του χρόνου συναρτήσει των dimensions διαφοροποιείται βραδύτερα. Αυτό φαίνεται άμα διαλέξουμε για παράδειγμα τις καμπύλες παραπάνω που αφορούν συγκεκριμένο αριθμό cluster (π.χ 1400 clusters – μαύρες καμπύλες) και παρατηρήσουμε για το εύρος τιμών για τα διαφορετικά πλήθη instances. Συγκεκριμένα για το παράδειγμα που αναφέραμε βλέπουμε ότι για 25000 instances έχουμε από 10 έως 1000 dimensions τιμές χρόνου από περίπου 100 sec μέχρι 5000 ενώ για τις περιπτώσεις 75000 και 100000 instances αυτό το εύρος είναι 280-20500 και 330-27200. Αυτό μεταφράζεται στο ότι σε μεγάλες τιμές πλήθους instances η διαφοροποίηση της χρονική απόδοσης σε διαφορετικές τιμές dimension τείνει να επικαλύπτεται από επίδραση που έχει στο χρόνο η αύξηση του πλήθους των instances.

#### 4.4.4.3 Επίδραση πλήθους συστάδων




*Σχήμα 4.5: Επίδραση χρόνου εκτέλεσης για διαφορετικό πλήθος cluster*

Αντίστοιχα με την ανάλυση που έχουμε ακολουθήσει μέχρι τώρα διαλέξαμε τα datasets με 25000,50000,100000 instances και κάναμε γραφικές για 3 διαφορετικά dimensions 50,250,750. Τα αποτελέσματα των γραφικών φαίνονται στο σχήμα 4.5 παραπάνω.

Επίσης παρατηρούμε μια γραμμική συμπεριφορά της εξάρτησης του χρόνου εκτέλεσης από το μέγεθος των cluster η οποία λόγω και της πιο πυκνής δειγματοληψίας μας για τις τιμές των cluster φαίνεται εντονότερα. Αντίστοιχα όπως και με την περίπτωση των διαφορετικών dimensions της προηγούμενης ενότητας μπορούμε και εδώ να πούμε ότι η επίδραση της αύξησης του μεγέθους των cluster επισκιάζεται όσο το πλήθος των instances αυξάνεται σημαντικά. Ένα ακόμα σημαντικό στοιχείο το οποίο γίνεται φανερό από τις παραπάνω γραφικές είναι ότι για μικρές τιμές dimension η αύξηση του μεγέθους των cluster έχει μεν αυξητική επίδραση στην χρονική εκτέλεση αλλά όχι τόσο έντονη όση αυτή που παρατηρείται για μεγαλύτερες τιμές του dimension.

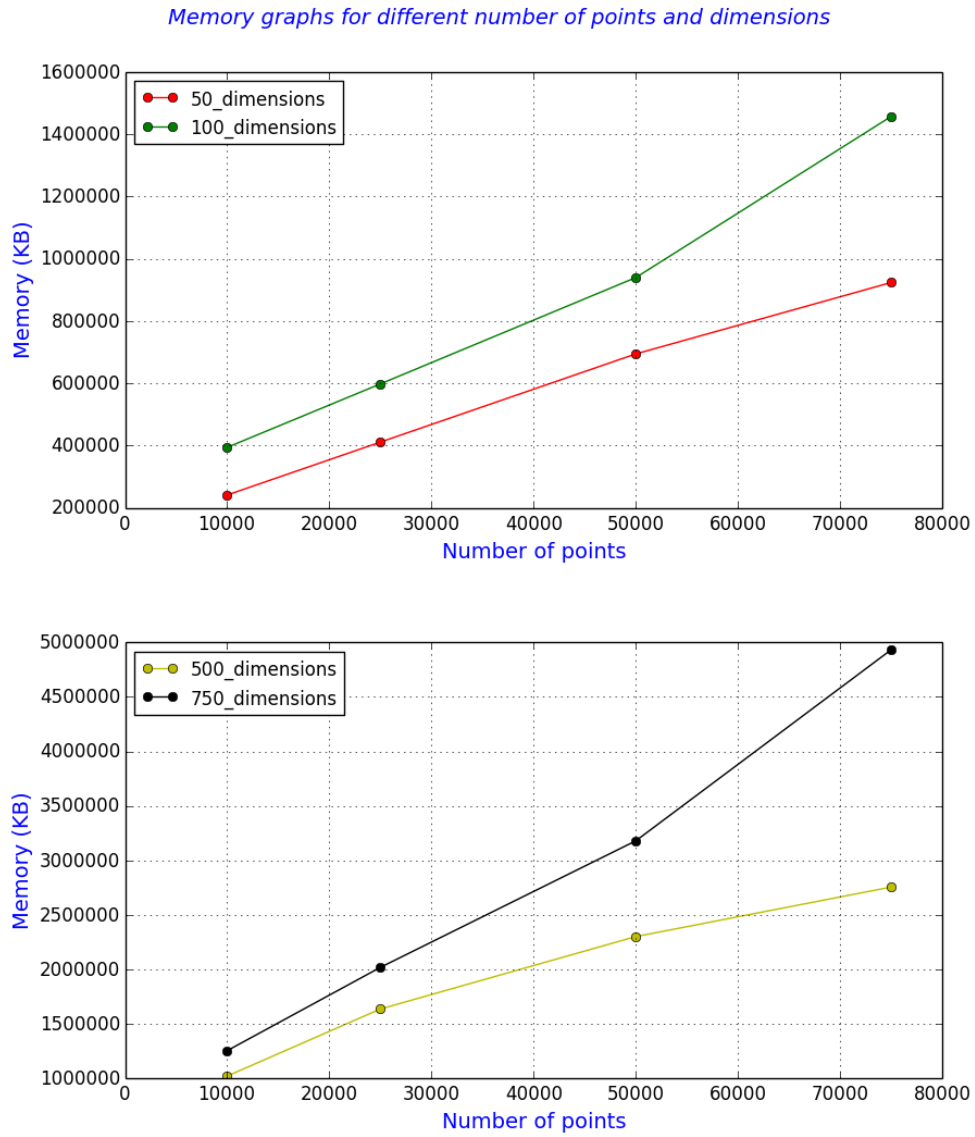
Κλείνοντας με αυτή την ενότητα την μελέτη μας σχετικά με την χρονική εξάρτηση της εκτέλεσης συναρτήσει των παραμέτρων πλήθους instances, dimensions και clusters, συγκεντρωτικά εδώ αναφέρουμε ότι η γραμμική εξάρτηση την οποία είδαμε σε όλες τις παραπάνω περιπτώσεις είναι και αυτή που θεωρητικά υποστηρίζεται από την χρονική πολυπλοκότητα του αλγορίθμου που είδαμε εισαγωγικά. Επαληθεύθηκε επίσης από τα γραφήματα που παραθέσαμε η βαρύτητα που έχει το πλήθος των instances το οποίο χαρακτηρίζει το dataset σε σχέση με τις άλλες δύο παραμέτρους, και αυτό διότι οι τιμές που λαμβάνει είναι συνήθως τάξεις μεγέθους μεγαλύτερες από τις υπόλοιπες.

## 4.4.5 Χρήση Μνήμης

Σε αυτή την υποενότητα παρουσιάζουμε τις μετρήσεις που πραγματοποιήσαμε σχετικά με το μέγεθος της χρησιμοποιούμενης μνήμης για κάθε εκτέλεση του αλγορίθμου. Γενικά παρατηρήθηκε ότι η μνήμη που χρησιμοποιήθηκε από τις εκτελέσεις του αλγορίθμου στο Weka επηρεάζεται και από το μέγεθος του πλήθους των instances αλλά και του πλήθους των dimension ενώ σε γενικές γραμμές διατηρεί μια σταθερή τιμή καθώς μεταβάλλουμε το πλήθος των cluster.

### 4.4.5.1 Επίδραση πλήθους instances

Για την εξάρτηση της χρήσης μνήμης από το πλήθος των instances χρησιμοποιούμε τα αποτελέσματα των μετρήσεων για τιμές των dimensions 50,100,500,750 και διαλέγουμε αριθμό clusters 800 αυθαίρετα καθώς όπως είπαμε στην αρχή αλλά και όπως θα δούμε στα επόμενα δεν υπάρχει ουσιαστική εξάρτηση του μεγέθους της μνήμης για διαφορετικές τιμές cluster. Τα παραπάνω φαίνονται στο Σχήμα 4.6 που παραθέτουμε εδώ:

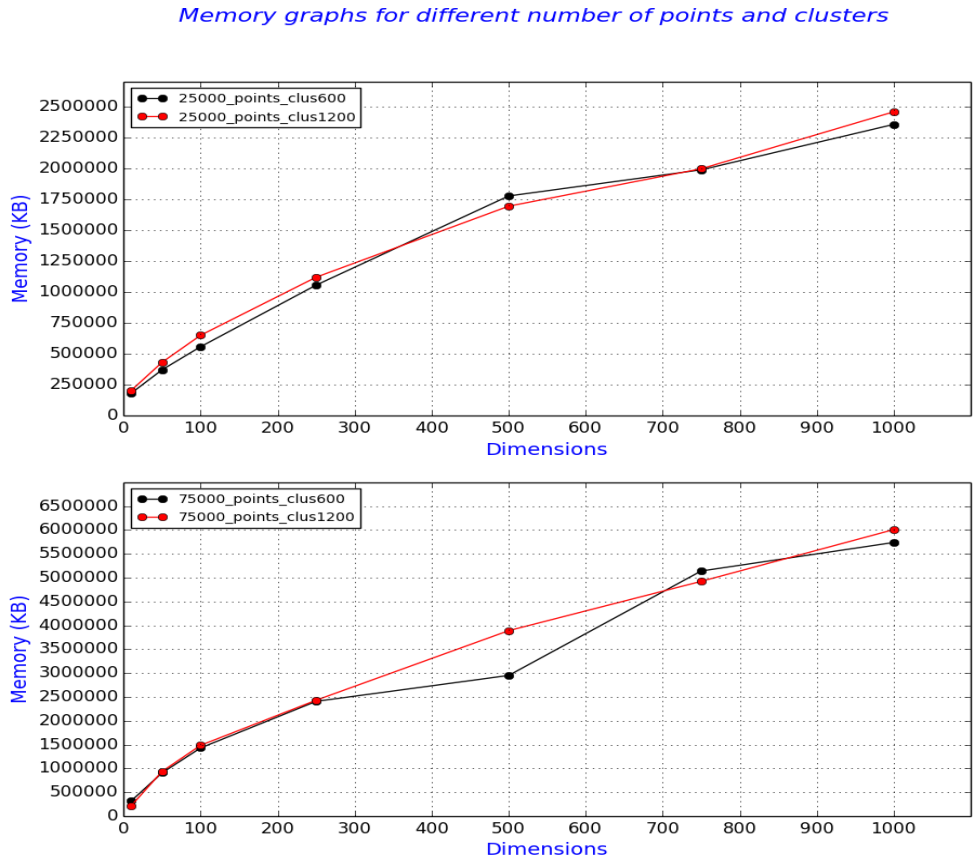



*Σχήμα 4.6: Επίδραση χρήσης μνήμης για διαφορετικά πλήθος instances*

Από τα παραπάνω γραφήματα βλέπουμε μια σχεδόν γραμμική εξάρτηση της μνήμης από το πλήθος των instances η οποία παρατηρείται ανεξάρτητα των διαφορετικών τιμών dimensions τις οποίες παρουσιάζουμε εδώ. Ένα ενδιαφέρον γεγονός που παρατηρείται από τα παραπάνω είναι ότι καθώς εναλλάσσουμε το μέγεθος των dimensions γίνεται ακόμα πιο αισθητή η αύξησης της χρησιμοποιούμενης μνήμης για κάθε διαφορετικό πλήθος instances. Αυτό επιβεβαιώνει και την διαισθητική μας αντίληψη ότι οι δύο αυτές παράμετροι είναι στενά συσχετιζόμενες όσο αναφορά την επίδραση στο μέγεθος της μνήμης.

### 4.4.5.2 Επίδραση πλήθους διαστάσεων

Όσο αναφορά την μνήμη σε σχέση με το πλήθος των διαστάσεων βλέπουμε αποτελέσματα μετρήσεων που κάναμε στο Σχήμα 4.7. Σε αυτό απεικονίζονται για δύο διαφορετικά πλήθη instances και συγκεκριμένα 25000 και 75000 και για δύο διαφορετικά cluster ανα περίπτωση οι τιμές μνήμης που μετρήσαμε.




*Σχήμα 4.7: Επίδραση χρήσης μνήμης για διαφορετικό πλήθος dimensions*

Όπως φαίνεται παρακάτω έχουμε και εδώ σχεδόν γραμμική εξάρτηση της χρησιμοποιούμενης μνήμης από το πλήθος των διαστάσεων του dataset και η παρατήρηση αυτή είναι ανεξάρτητη τόσο του πλήθους των instances όσο και του μεγέθους των διαφορετικών cluster. Αυτό που ίσως αξίξει να επισημάνουμε εδώ είναι είναι ότι η υλοποίηση του Weka κάνει υπερβολική κατανάλωση κύριας μνήμης σε σύγκριση με το μέγεθος του dataset εισόδου. Αυτό διαφαίνεται καλύτερα στον Πίνακα 4.3 για την περίπτωση των 75000 instances και για τα διαφορετικά dimensions.

| Dataset size (MB) per dimension | Memory used (MB) |
|---|---|
| **15** | **273** |
| **75** | **840** |
| **150** | **1332** |
| **350** | **2345** |

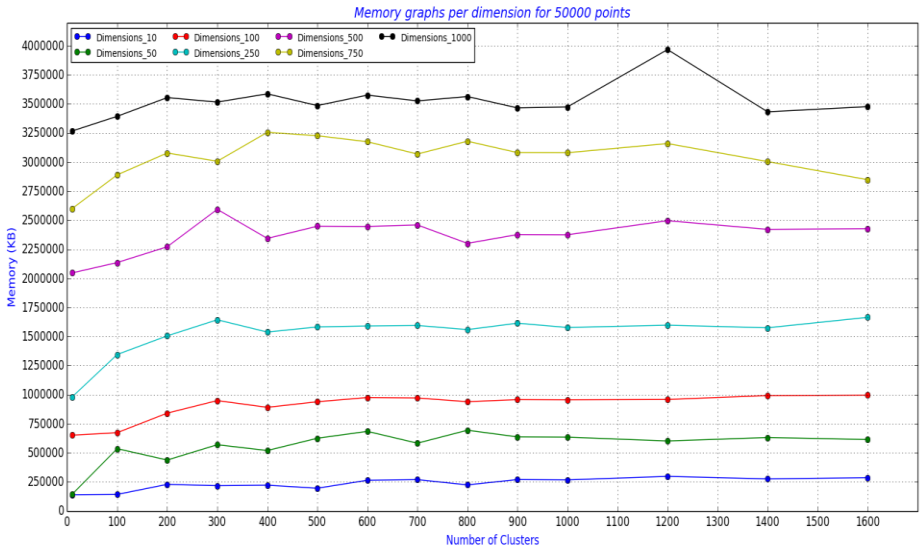
| | |
|---|---|
| **750** | **3384** |
| **1100** | **4704** |
| **1500** | **5771** |

*Πίνακας 4.3: Χρήση μνήμης ανά μέγεθος dataset*

Στο παραπάνω πίνακα η χρήση μνήμης είναι ο μέσος όρος των μετρήσεων όλων των εκτελέσεων για κάθε cluster (14 συνολικά). Είναι φανερό ότι η υλοποίηση στο Weka είναι ιδιαίτερα σπάταλη στις ανάγκες τις για μνήμη και αυτό ίσως είναι ένα γεγονός που να υποδεικνύει ότι ο κώδικας που υλοποιεί τον αλγόριθμο επιδέχεται βελτιώσεις στο τρόπο που διαχειρίζεται την μνήμη.

### 4.4.5.3 Επίδραση πλήθους συστάδων

Η επίδραση του πλήθους των συστάδων για κάθε εκτέλεση όπως μπορούμε διαισθητικά να προβλέψουμε δεν θα έπρεπε να έχει επίπτωση στην μετρούμενη χρήση μνήμης. Αυτό γιατί σε κάθε επανάληψη του αλγορίθμου χρειαζόμαστε να έχουμε αποθηκευμένα μία μόνο φορά τα κεντροειδή που σχετίζονται με την εκάστοτε επανάληψη. Αυτά σε σύγκριση με το συνολικό μέγεθος του dataset που θέλουμε να επίσης να έχουμε διαθέσιμο στη μνήμη κάθε στιγμή είναι αμελητέο μέγεθος. Τα παραπάνω επιβεβαιώνονται και στο Σχήμα 4.8 παρακάτω που αφορά την εκτέλεση για 50000 instances και για όλα τα πιθανά dimensions.




*Σχήμα 4.8: Επίδραση χρήσης μνήμης για διαφορετικό πλήθος cluster*

## 4.5 Μοντελοποίηση

Η ανάλυση των προηγούμενων ενοτήτων εκτός από τις χρήσιμες πληροφορίες που μας παρείχε σχετικά με την κεντρική υλοποίηση του αλγορίθμου, αποτελεί επίσης και οδηγό μας, για τη διαδικασία της μοντελοποίησης που θα ακολουθήσουμε κυρίως όσο αναφορά το ποιες από τις μετρικές αξίζει να μοντελοποιηθεί η συμπεριφορά τους. Αναλυτικότερα, με αυτά που είδαμε στις προηγούμενες ενότητες είναι φανερό ότι αξία μοντελοποίησης παρέχουν αρχικά η χρονική επίδοση για κάθε εκτέλεση καθώς και η χρησιμοποιούμενη μνήμη ως προς κάθε μεταβαλλόμενη μετρική.

Χωρίζουμε λοιπόν την μοντελοποίηση σε δύο σκέλη. Το πρώτο αφορά μοντέλα που κατασκευάζουμε/εκπαιδεύουμε αποκλειστικά για την χρονική επίδοση του αλγορίθμου, ενώ το δεύτερο αφορά μοντέλα τα οποία κατασκευάστηκαν για να περιγράψουν την χρήση μνήμης για κάθε εκτέλεση. Ακόμα πιο σημαντικό από την εκπαίδευση των μοντέλων είναι να παρουσιάσουμε την ακρίβεια αυτών των μοντέλων σε σχέση με τις πραγματικές τιμές/μετρήσεις που έχουμε και να να τα συγκρίνουμε μεταξύ τους ώστε να καταλήξουμε ποιο είναι αυτό που εκφράζει καλύτερα στην ολότητά της την υλοποίηση. Για τους σκοπούς αυτούς αφιερώνονται οι επόμενες δύο υποενότητες.

### 4.5.1 Μοντελοποίηση χρόνου εκτέλεσης

Πριν προχωρήσουμε στη παρουσίαση των μοντέλων που κατασκευάσαμε, είναι σκόπιμο εδώ να παραθέσουμε ένα δείγμα των δεδομένων εισόδου που αποτελούν ουσιαστικά το σύνολο εκπαίδευσης με το οποίο δουλεύουμε. Στο πίνακα 4.4 δείχνουμε πέντε διαφορετικά σημεία (data points). Οι πρώτες τρεις τιμές αποτελούν τις τιμές εισόδου, ενώ η τελευταία που αφορά τον χρόνο είναι η τιμή ενδιαφέροντος που προσπαθούμε να εκτιμήσουμε/προσεγγίσουμε με τα μοντέλα που θα παρουσιάσουμε, έχοντας ως πρότερη γνώση αυτής της τιμής τα πειράματα που έχουμε εκτελέσει. Συνολικά τα παραπάνω data points είναι 490 όσες δηλαδή και οι εκτελέσεις που κάναμε για την κεντρική υλοποίηση.

| Instances | Dimensions | k (clusters) | Execution_time (sec) |
|---|---|---|---|
| 10000 | 10 | 200 | 7 |
| 25000 | 50 | 400 | 101 |
| 50000 | 100 | 600 | 671 |
| 75000 | 250 | 800 | 8799 |
| 100000 | 500 | 1400 | 11876 |

*Πίνακας 4.4: Δείγμα δεδομένων μοντελοποίησης χρόνου εκτέλεσης*

Αρχικός σκοπός μας ήταν με το σύνολο αυτών των data points να κατασκευάσουμε καθολικά μοντέλα τα οποία να προσεγγίζουν την χρονική συμπεριφορά εκτέλεσης του αλγορίθμου. Η ακρίβεια προσέγγισης ενός μοντέλου επηρεάζεται από το σύνολο και το εύρος τιμών της τιμής που προσπαθούμε να εκτιμήσουμε. Συγκεκριμένα αν το σύνολο των δεδομένων εισόδου εκφράζει διαφορετικές συμπεριφορές του φυσικού

φαινομένου/πειράματος τόσο πιο δύσκολο είναι αυτές οι συμπεριφορές να προσεγγιστούν από ένα μόνο μοντέλο.

Το παραπάνω φαινόμενο παρατηρήθηκε ακριβώς και στη περίπτωσή μας όπου παρατηρούμε μία απότομη αύξηση της χρονικής εκτέλεσης καθώς αυξάνουμε το πλήθος των instances, των dimension και των cluster σε σύγκριση με μικρότερες τιμές αυτών. Η παραπάνω παρατήρηση μας οδήγησε να διαχωρίσουμε το σύνολο των data points σε δύο κατηγορίες. Η μία αφορά σε στιγμιότυπα εκτέλεσης με χρόνο μικρότερο των 1000 sec και η δεύτερη σε στιγμιότυπα με χρόνο μεγαλύτερο των 1000 sec. Αυτή η επιλογή έγινε διότι παρατηρήσαμε ότι υπάρχει μια συγκέντρωση τιμών μεγαλύτερη του 50% του συνολικού dataset στη πρώτη κατηγορία (280/490 datapoints) και τα υπόλοιπα instances εμπίπτουν στην δεύτερη με ένα εύρος τιμών από 1000 sec μέχρι 31000 sec περίπου. Έτσι λοιπόν με αυτό το όριο στην τιμή του χρόνου επιτυγχάνουμε να απομονώσουμε τις δύο διαφορετικές συμπεριφορές και να επιτύχουμε υψηλότερη ακρίβεια για τα μοντέλα μας.

Στα επόμενα θα παρουσιάσουμε λοιπόν μοντέλα που κατασκευάσαμε με την χρήση των data points με χρόνο εκτέλεσης μικρότερο των 1000 sec. Επισημαίνουμε παρόλα αυτά ότι τα ίδια μοντέλα που θα δούμε εδώ μπορούν να χρησιμοποιηθούν και για την μοντελοποίηση της δεύτερης κατηγορίας με χρονική εκτέλεση μεγαλύτερη των 1000 sec, απλά εδώ θα αποφύγουμε να το παρουσιάσουμε για να μην επεκταθούμε άσκοπα.

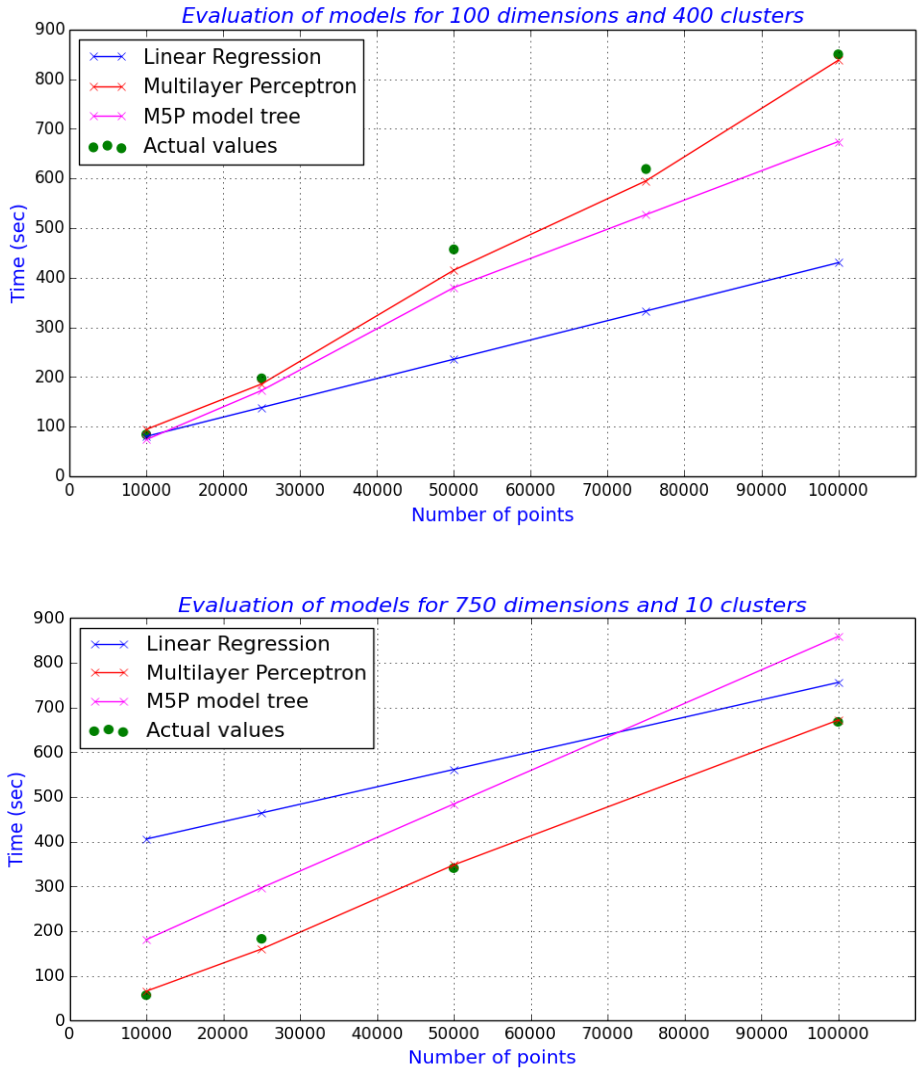
Για την κατασκευή των μοντέλων χρησιμοποιήσαμε το Weka όπως είχαμε αναφέρει και εισαγωγικά και συγκεκριμένα για την μοντελοποίηση χρονικής εκτέλεσης δοκιμάσαμε να κατασκευάσουμε ένα μοντέλο με linear regression, ένα με Multilayer Perceptron και ένα ακόμα με M5P model tree. Το Weka δίνει τη δυνατότητα το αρχικά οριζόμενο dataset να διασπαστεί αυτομάτως σε training και test σετ. Εμείς επιλέξαμε από τα 280 data points που χρησιμοποιούμε για την μοντελοποίηση, να διατηρήσουμε ένα 10% (28 data points) για την επαλήθευση του κάθε μοντέλου και ένα 90% για την εκπαίδευση του μοντέλου. Στο Πίνακα 4.5 παραθέτουμε τα mean absolute error και mean relative error που πήραμε ως αποτέλεσμα από την επαλήθευση του κάθε μοντέλου με το αντίστοιχο training set.

| Models | Mean Absolute Error | Mean Relative Error |
|---|---|---|
| Linear Regression | 111.97 | 51.58% |
| Multilayer Perceptron | 70.57 | 32.51% |
| M5P tree model | 58.61 | 27% |

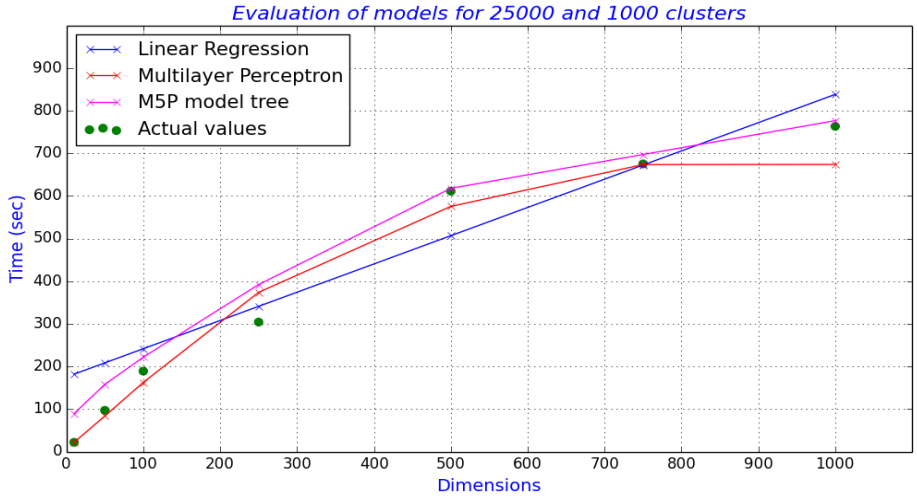
*Πίνακας 4.5: Σφάλματα μοντελοποίησης χρόνου εκτέλεσης*

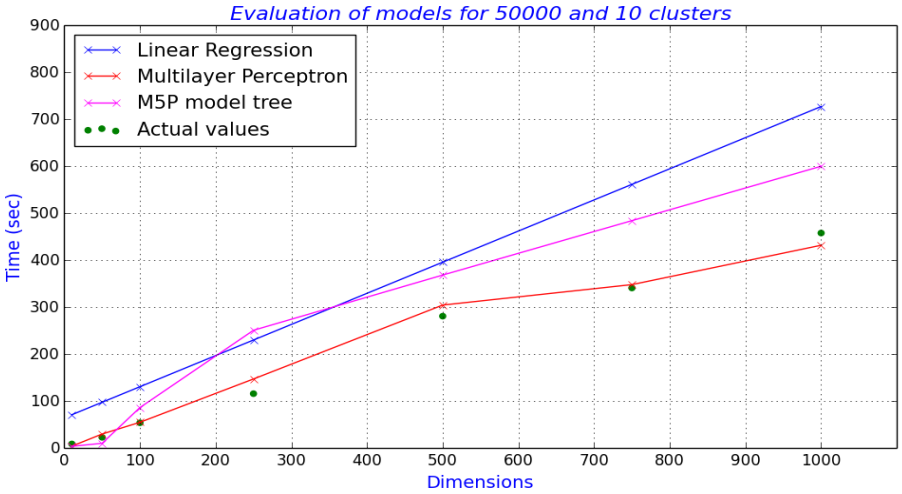
Ο παραπάνω πίνακας μας δίνει μία πρώτη εντύπωση της ποιότητας των μοντέλων μέσω του μέσου απόλυτου σφάλματος (mean absolute error) αλλά και του μέσου σχετικού σφάλματος (mean relative error) παρόλα αυτά επειδή η επαλήθευση δεν γίνεται επί του ίδιου ακριβώς training set αυτές οι τιμές από μόνες τους δεν μπορούν να μας οδηγήσουν σε ασφαλή συγκριτικά συμπεράσματα. Εκτός αυτού και λόγω επίσης του πολυδιάστατου χώρου που εφράζουν τα data points εισόδου έχει ενδιαφέρον να δούμε πως το κάθε μοντέλο εκτιμά την χρονική εκτέλεση του κάθε πειράματος για τη μεταβολή κάθε φορά μόνο μιας παραμέτρου εισόδου.

Για τον παραπάνω λόγω επανατροφοδοτήσαμε τα όλα τα data points στα μοντέλα που κατασκευάσαμε παραπάνω και λάβαμε τις εκτιμήσεις για κάθε τιμή εισόδου. Με αυτά τα αποτελέσματα φτιάξαμε γραφικές που παρουσιάζουν πως παρεμβάλλονται οι εκτιμήσεις αυτές ανάμεσα στις πραγματικές τιμές. Στα Σχήματα 4.9 έως 4.11 επιλέγουμε να διατηρούμε σταθερές τις δύο από τις παραμέτρους εισόδου για διαφορετικές τιμές τους και να μεταβάλλουμε την εναπομένουσα.

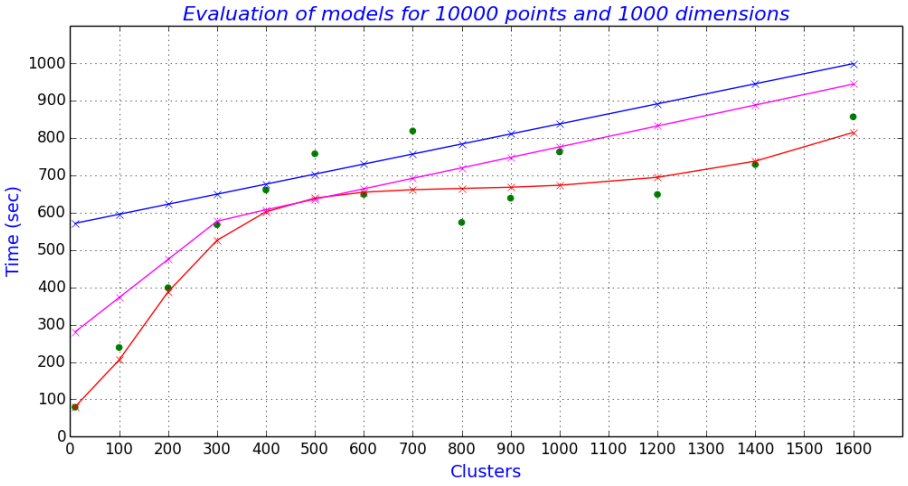


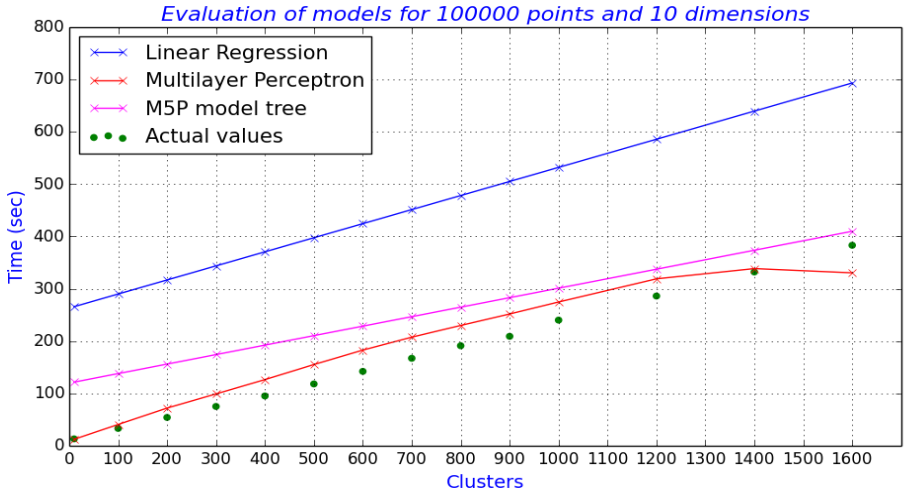

*Σχήμα 4.9: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο αριθμό instances*




*Σχήμα 4.10: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο αριθμό dimensions*






*Σχήμα 4.11: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο αριθμό cluster*

Τα παραπάνω γραφήματα μας δίνουν μια πληρέστερη εικόνα της επίδοσης των μοντέλων που κατασκευάσαμε. Συγκεκριμένα παρατηρούμε ότι το linear regression μοντέλο που επιχειρεί να βρεί γραμμική συσχέτιση της εξόδου με τις μεταβλητές εισόδου έχει κακή απόδοση, ειδικά στις περιπτώσεις όπου μεταβάλλουμε μόνο τον αριθμό τον αριθμό των instances και τον αριθμό των cluster. Το tree model μαζί με το multilayer perceptron νευρωνικό δίκτυο είναι αυτά που γενικά επιτυγχάνουν καλύτερη σύγκλιση σε όλες τις περιπτώσεις. Αυτό που φαίνεται για το model tree ειδικότερα, είναι ότι καθώς αυξάνουμε τις παραμέτρους του πλήθους των instances αλλά και των dimension αποκλίνει περισσότερο συγκρινόμενο με μικρότερες τιμές των ίδιων παραμέτρων. Η γενική εικόνα που αποκομίζουμε είναι ότι το νευρωνικό δίκτυο προσεγγίζει σε πολύ ικανοποιητικό βαθμό την χρονική επίδοση της κεντρικής υλοποίησης για κάθε μεταβαλλόμενη μεταβλητή.

Ένα ερώτημα που εγείρεται είναι, γιατί αποτυγχάνει σε τέτοιο βαθμό η γραμμική μοντελοποίηση εφ'όσον σε όλη τη προηγούμενη ανάλυσή μας είδαμε σχεδόν παντού γραμμική συσχέτιση της χρονική επίδοσης για διαφορετικές περιπτώσεις. Η απάντηση είναι ότι στις περιπτώσεις που παρατηρήσαμε κάτι τέτοιο, σταθεροποιούσαμε 2 μεταβλητές εκτέλεσεις (λχ dimensions και clusters) και μεταβάλλαμε την έταιρη και στο χώρο εκείνο βλέπαμε γραμμική συμπεριφορά. Στη μοντελοποίηση που επιχειρούμε εδώ το γραμμικό μοντέλο προσπαθεί να παρεμβληθεί σε διαφορετικό χώρο διαστάσεων όπου εξαλείφεται αυτή η γραμμική εξάρτηση από ότι φαίνεται. Εκεί ακριβώς φαίνεται και η εν μέρει επιτυχία του μοντέλου M5P tree όπου αφού κατασκευάζεται ένα δέντρο απόφασης για τα data points τότε στα φύλλα του δέντρου αυτού κατασκευάζονται γραμμικά μοντέλα που δίνουν την τελική εκτίμηση τιμής. Αυτό ισοδύναμα μεταφράζεται στο ότι το tree model επιχειρεί να βρει γραμμική εξάρτηση σε υποπεριοχές του αρχικού χώρου, και επειδή εδώ έχουμε ακριβώς μία τέτοια συμπεριφορά, καταφέρνει να πετύχει μια αρκετά καλή προσέγγιση. Αυτό που όμως κρατάμε από την παραπάνω ανάλυση είναι η μεγάλη εκφραστική δύναμη του νευρωνικού δικτύου που φαίνεται από την ικανότητα του να παρεμβάλλεται με πολύ μεγάλη ακρίβεια στο χώρο για τον οποίο εκπαιδεύεται.

### 4.5.2 Μοντελοποίηση χρήσης μνήμης

Σχετικά με τη μοντελοποίηση για την χρήση μνήμης για κάθε εκτέλεση, όπως και στη προηγούμενη παράγραφο παρουσιάζουμε στον Πίνακα 4.6 παρακάτω ένα δείγμα των δεδομένων εισόδου για το κάθε μοντέλο που κατασκευάσαμε.

| Instances | Dimensions | k (clusters) | Mem_used (KB) |
|---|---|---|---|
| 10000 | 10 | 200 | 191564 |
| 25000 | 50 | 400 | 427299 |
| 50000 | 100 | 600 | 975086 |
| 75000 | 250 | 800 | 2499516 |
| 100000 | 500 | 1400 | 3507273 |

*Πίνακας 4.6: Δείγμα δεδομένων μοντελοποίησης χρήσης μνήμης*

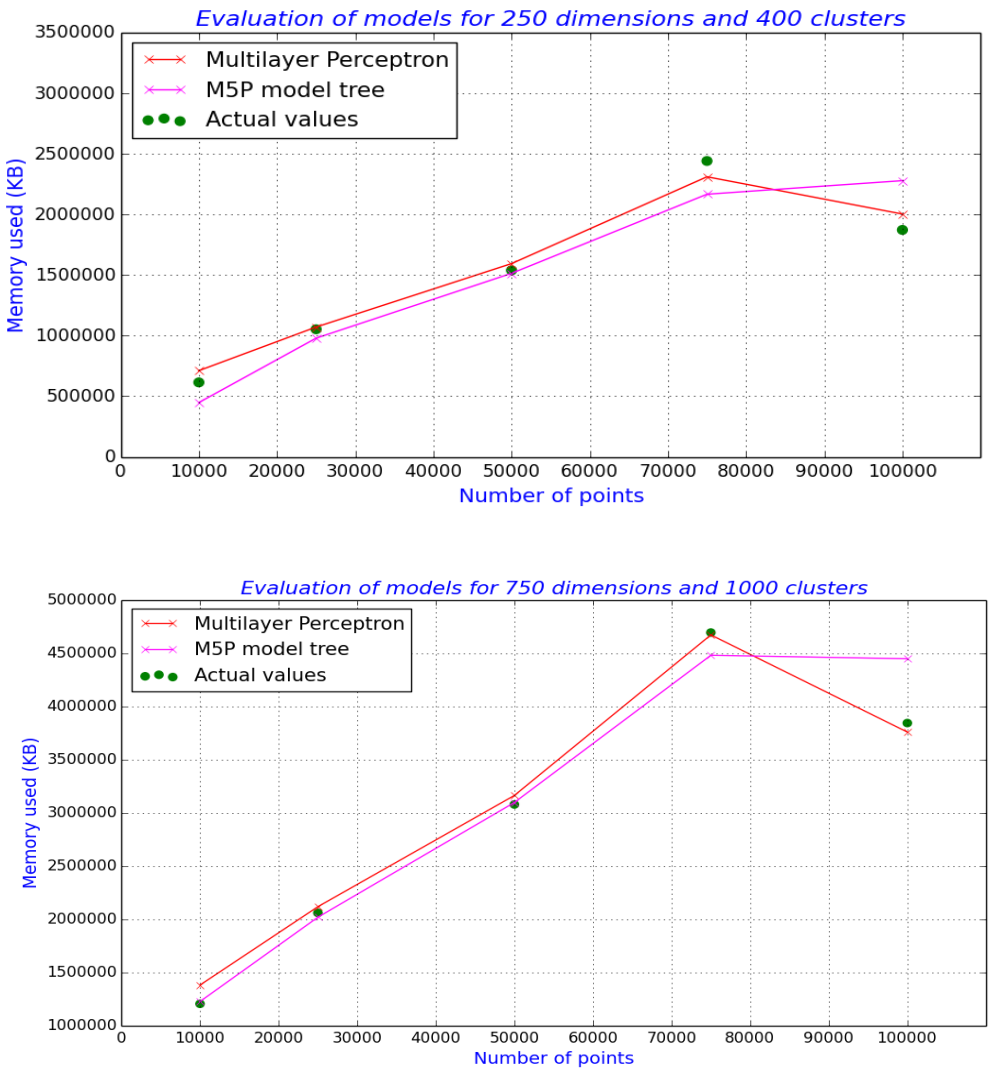
Σε αντίθεση όμως με την ανάγκη διαχωρισμού των data points που περιγράψαμε για την περίπτωση της χρονικής εκτέλεσης, το εύρος τιμών της χρησιμοποιούμενης κύριας μνήμης είναι πολύ μικρότερο και η κατασκευή ενός ενιαίου μοντέλου που να προσεγγίζει όλες τις τιμές του χώρου είναι εφικτό. Για αυτό το λόγο για την κατασκευή των μοντέλων χρησιμοποιούμε όλα τα αποτελέσματά μας από τις εκτελέσεις για την κεντρική υλοποίηση (490 συνολικά) κάνοντας πάλι ένα διαχωρισμό αυτών σε training set (90% του συνόλου) και σε test set (10% του συνόλου), ώστε με το τελευταίο να ελέγξουμε την ακρίβεια μοντελοποίησης. Όσο αναφορά τα μοντέλα που χρησιμοποιήσαμε παραμένουν και αυτά ίδια με την προηγούμενη ενότητα δηλαδή ένα Multilayer perceptron, ένα linear regression και ένα M5P tree model. Συγκεντρώνουμε τα σφάλματα που παρατηρήσαμε για κάθε περίπτωση στον Πίνακα 4.4.

| Models | Mean Absolute Error | Mean Relative Error |
|---|---|---|
| Linear Regression | 701662 | 50.29% |
| Multilayer Perceptron | 191916 | 13.75% |
| M5P tree model | 187614 | 13.45% |

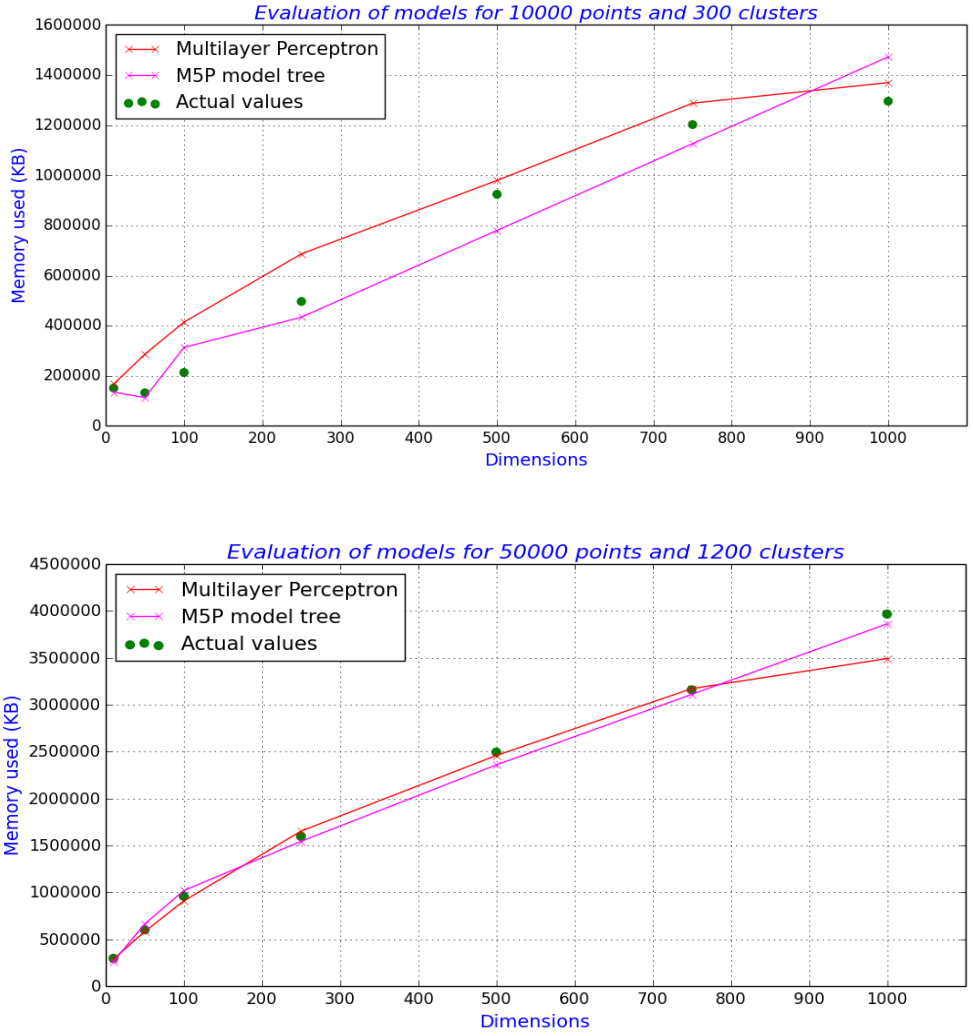
*Πίνακας 4.7: Σφάλματα μοντελοποίησης χρήσης μνήμης*

Από τα αποτελέσματα για τα σφάλματα που προέκυψαν του Πίνακα 4.7 βλέπουμε ότι το γραμμικό μοντέλο παρουσιάζει μεγάλο μέσο απόλυτο σφάλμα το οποίο σε σύγκριση με τις πραγματικές τιμές του training set αντιστοιχεί και σε σημαντικό μέσο σχετικό σφάλμα. Αντίθετα τα το νευρωνικό δίκτυο και το tree model έχουν πολύ καλύτερη προσέγγιση που μεταφράζεται σε παρόμοιες τιμές και για τα δύο προαναφερθέντα σφάλματα. Επειδή παρατηρείται αυτή η διαφορά αποφεύγουμε στα επόμενα να παρουσιάσουμε γραφικά το linear regression μοντέλο και επικεντρωνόμαστε στα τελευταία δύο μοντέλα.

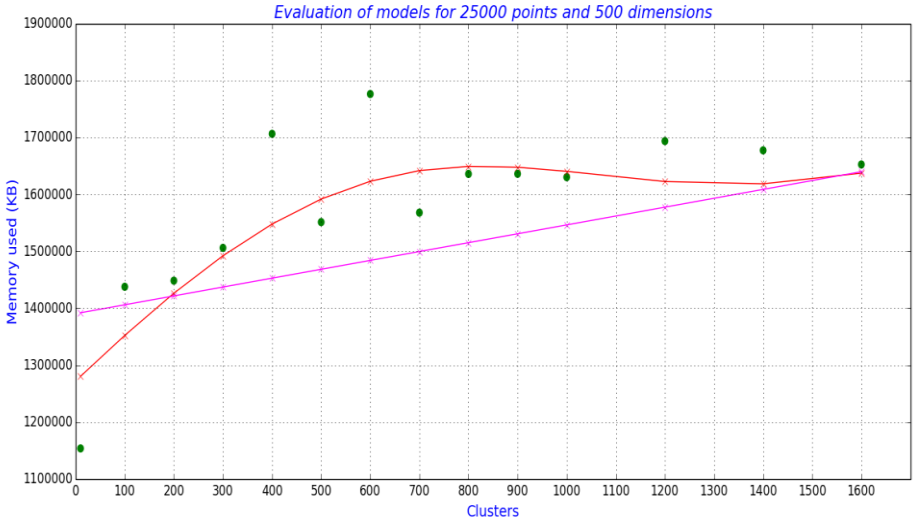
Στα Σχήματα 4.12 – 4.14 λοιπόν παρουσιάζουμε γραφικά τις παρεμβληθείσες τιμές των μοντέλων του νευρωνικού δικτύου και του M5P tree model με τις δύο από τις παραμέτρους σταθερές και την έταιρη να μεταβάλλεται.Σκοπός είναι όπως και πριν να δούμε πως ανταποκρίνονται τα μοντέλα μας σε κάθε περίπτωση.

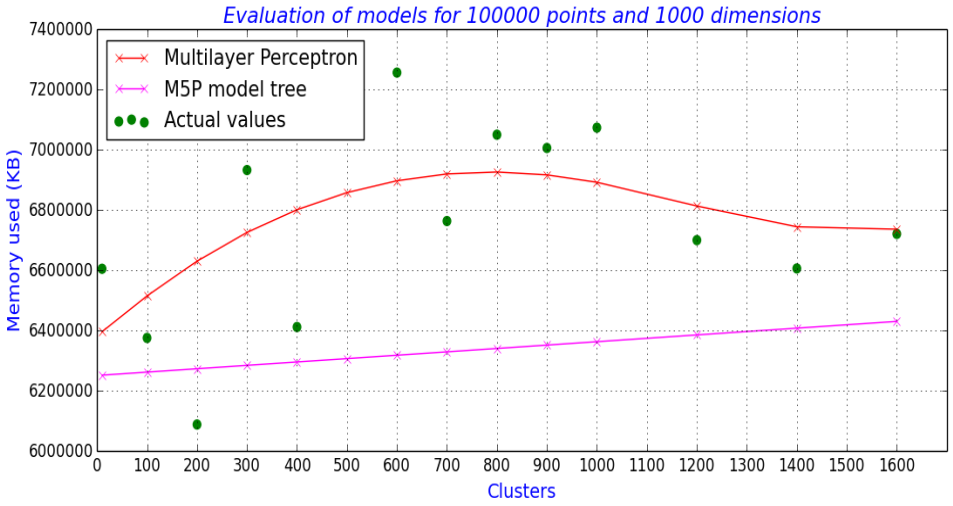



*Σχήμα 4.12: Μοντελοποίηση χρήσης μνήμης για μεταβαλλόμενο αριθμό instances*




*Σχήμα 4.13: Μοντελοποίηση χρήσης μνήμης για μεταβαλλόμενο αριθμό dimensions*






*Σχήμα 4.14: Μοντελοποίηση χρήσης μνήμης για μεταβαλλόμενο αριθμό cluster*

Τα παραπάνω γραφήματα αναδεικνύουν την επιτυχία και των δύο μοντέλων στο να περιγράψουν με μεγάλη ακρίβεια την χρησιμοποιούμενη μνήμη σχεδόν για κάθε περίπτωση. Κάτι που ίσως αξίζει να σημειώσουμε φαίνεται στα πρώτα 2 γραφήματα που αφορούν τη μεταβολή του αριθμού των instances για συγκεκριμένα cluster και dimension. Συγκεκριμένα η πτώση της τιμής της μνήμης από τα 75000 στα 100000 instances είναι μια μη αναμενόμενη εξέλιξη που ήδη κάνει σχετική αναφορά στην υποενότητα 4.4.5.1. Σε αυτή τη μεταβολή από ότι φαίνεται το νευρωνικό δίκτυο παρουσιάζεται πιο 'ευαίσθητο' ακολουθώντας κατά μία έννοια την παραβατική αυτή συμπεριφορά ενώ αντίθετα για το tree model παρατηρούμε ότι επιδεικνύει μια μεγαλύτερη αδράνεια σε τέτοιου είδους θορύβους. Το τελευταίο είναι ένα σημαντικό πλεονέκτημα που επιδεικνύει στην παρούσα κατάσταση το tree model, ιδιαίτερα για την περίπτωση της μελέτης μας όπου το ρίσκο να εμπεριέχονται μέσα στις μετρήσεις μας παραβατικές συμπεριφορές είναι ιδιαίτερα υψηλό λόγω πολλών και μη διαχειρίσιμων λόγων που σχετίζονται με το περιβάλλον εκτέλεσης.

Ακόμα ένα στοιχείο που αξίζει να επισημάνουμε είναι η σημαντική διαφορά μεταξύ των μοντέλων στο Σχήμα 4.14 για τις γραφικές όπου μεταβάλλουμε μόνο το μέγεθος των cluster. Εδώ φαίνεται ότι η ακρίβεια του νευρωνικού δικτύου υπερτερεί του tree model, παρόλα αυτά επειδή όπως έχουμε δει στην ενότητα 4.4.5.3 η μνήμη παραμένει γενικά ανεπηρέαστη από την παράμετρο των cluster με μικρές σχετικά αποκλίσεις η αδυναμία του tree model σε αυτή τη περίπτωση δεν είναι βαρύνουσα.

Συνοψίζοντας τα παραπάνω είναι θετική η απόδοση και των δύο παρουσιαζόμενων μοντέλων αλλά η μικρή διαφορά που έχουν μαζί με το γεγονός ότι το tree model από ότι φαίνεται παρουσιάζει μία μεγαλύτερη ανοχή στα σφάλματα μετρήσεων καθιστούν το τελευταίο ιδανικότερο μοντέλο, στα πλαίσια της παρούσας μελέτης.

# Κεφάλαιο 5

# Εκτέλεση και μοντελοποίηση του αλγορίθμου K-means στο Apache Spark

Το κεφάλαιο αυτό αφορά την εκτέλεση του αλγορίθμου k-means κατανεμημένα στο Apache Spark. Στις πρώτες δύο ενότητες θα παρουσιάσουμε αρχικά την πειραματική διαδικασία που ακολουθήσαμε για τη μελέτη μας, καθώς και τον τρόπο που το Spark εκτελεί τον αλγόριθμο σε ένα cluster υπολογιστών. Στη συνέχεια θα παρουσιάσουμε και θα αναλύσουμε τα αποτελέσματα των μετρήσεών μας για τις διαφορετικές περιπτώσεις εκτέλεσης. Έπειτα αφιερώνουμε δύο ακόμα ενότητες που αφορούν στη σύγκριση των εκτελέσεων αρχικά με διαφορετικό μέγεθος cluster και επίσης σε σχέση με την κεντρική υπολοποίηση σε Weka που ήδη έχουμε δει. Η τελευταία ενότητα του κεφαλαίου θα επικεντρωθεί στη μοντελοποίηση της κατανεμημένης υλοποίησης και της μελέτης της ακρίβειας των κατασκευασθέντων μοντέλων.

## 5.1 Περιγραφή πειραματικής διαδικασίας

Ο σκοπός της μελέτης μας παραμένει κατά βάση ο ίδιος με αυτόν που περιγράψαμε στην ενότητα 4.1 για την κεντρική υλοποίηση. Συγκεκριμένα μας ενδιαφέρει να μελετήσουμε την εκτέλεση του αλγορίθμου k-means για διαφορετικά dataset (πλήθος instances, dimensions), διαφορετικές παραμέτρους εκτέλεσης (clusters,iterations) και διαφορετικό περιβάλλον εκτέλεσης. Η μεγαλύτερη διαφορά σε σχέση με την κεντρική υλοποίηση αφορά στην εναλλαγή του περιβάλλοντος εκτέλεσης, που έχει να κάνει με τον αριθμό των κόμβων (nodes) που αποτελούν το cluster, το μέγεθος της μνήμης ανά node και το πλήθος των cores ανά node. Όπως είναι αναμενόμενο οι παραπάνω παράγοντες επηρεάζουν ευθέως την συμπεριφορά εκτέλεσης ως προς τις μετρήσεις που θέλουμε να κάνουμε σχετικά με το χρόνο εκτέλεσης, τη συνολική χρησιμοποιούμενη μνήμη του cluster και το disk input και output ενώ παράλληλα προσθέτουν και μεγαλύτερη πολυπλοκότητα στο εγχείρημα της κατανεμημένης μελέτης.

Όπως και στην κεντρική υλοποίηση η προσέγγισή μας πρέπει να συγκεράσει την διαχείριση της πολυπλοκότητας του χώρου πειραμάτων που προκύπτει εναλλάσσοντας όλες τις παραπάνω παραμέτρους με την εξαγωγή αντιπροσωπευτικών αποτελεσμάτων από τις εκτελέσεις που θα επιλέξουμε. Εξάλλου μας ενδιαφέρει επίσης και η σύγκριση της κατανεμημένης με την κεντρική υλοποίηση στο μέγεθος που αυτή είναι δυνατή. Θέτοντας τα παραπάνω ως στόχους επιλέξαμε να εκτελέσουμε ένα σύνολο πειραμάτων για τους δύο διαφορετικούς τύπους cluster που φαίνονται στο Πίνακα 5.1 και για τα datasets του Πίνακα 5.2.

| **Τύπος cluster** | **Πλήθος nodes ανά cluster** | **Πλήθος cores ανά node** | **Μέγεθος μνήμης ανά node** |
|---|---|---|---|
| Τύπος I | 1 master<br>4 workers | 4 cores (master)<br>2 cores (workers) | 8GB (master)<br>4 GB (workers) |
| Τύπος II | 1 master<br>8 workers | 4 cores (master)<br>2 cores (workers) | 8 GB (master)<br>8 GB (workers) |

*Πίνακας 5.1: Τύποι cluster για κατανεμημένη εκτέλεση*

| **Τιμές πλήθους instances** | **Τιμές πλήθους dimensions** | **Τιμές Clusters (K)** |
|---|---|---|
| 75000,100000,150000 | 1000,2000,3000,<br>4000,5000,6000 | 100,250,500,750,1000 |

*Πίνακας 5.2: Τιμές instances,dimensions και cluster υλοποίησης Spark*

Το πλήθος των cores ανά node ορίζει ουσιαστικά το πόσα task μπορεί να εκτελεί ταυτόχρονα ο κάθε κόμβος, οπότε επιλέξαμε για τα δύο διαφορετικά cluster να διατηρήσουμε την υπολογιστική ικανότητα του κάθε κόμβου σταθερή. Επίσης σταθερή επιλέξαμε να διατηρήσουμε και τη συνολική μνήμη που διαθέτει και στις δύο περιπτώσεις το cluster, αφού αυξήσαμε τους κόμβους σε διπλάσιο αριθμό αλλά υποδιπλασιάσαμε το μέγεθος της μνήμης ανά node. Οι παραπάνω δύο επιλογές έγιναν με γνώμονα να παρατηρήσουμε πως κλιμακώνεται η απόδοση της εκτέλεσης του αλγορίθμου από τη διαφοροποίηση του πλήθους των nodes αποκλειστικά, για να καταλήξουμε σε συγκριτικά συμπεράσματα στα οποία θα αναφερθούμε σε μεταγενέστερη ενότητα.

Για τα datasets που χρησιμοποιήθηκαν για να γίνουν οι εκτελέσεις και στα δύο cluster, επιλέχθηκαν οι τιμές που φαίνονται στον Πίνακα 5.2 σχετικά με το πλήθος των instances, τα dimensions αλλά και τις τιμές των cluster για κάθε εκτέλεση. Για την επιλογή των τιμών επιλέξαμε ομοιόμορφη δειγματοληψία εκτός λίγων εξαιρέσεων, παράλληλα όμως περιοριστήκαμε και από τις ικανότητες του Spark μετά από δοκιμαστικές εκτελέσεις. Συγκεκριμένα παρατηρήσαμε ότι στην περίπτωση και των δύο cluster, η εκτέλεση με 150000 instances, 6000 dimensions και 1000 clusters αδυνατούσε να τερματίσει κανονικά λόγω ανεπάρκειας μνήμης του cluster μετά από διαδοχικές δοκιμές. Το παραπάνω γεγονός μας έθεσε ένα ανώτατο όριο μεγέθους του dataset που μπορούμε να χρησιμοποιήσουμε και άρα ένα αντίστοιχο περιορισμό όσο αναφορά τις τιμές των instances,dimensions αλλά και των clusters.

Οι συνολικές λοιπόν εκτελέσεις που πραγματοποιήσαμε και για τα δύο είδη cluster υπολογίζονται:

**Εκτελέσεις = 2 (type of clusters) x 3 (#instances) x 6 (dimensions) x 5 (clusters) = 180**

Από αυτές οι 90 που αφορούν το cluster με τους 9 nodes συνολικά, χρησιμοποιήθηκαν τόσο για την ανάλυση της συμπεριφοράς εκτέλεσης αλλά και για την μοντελοποίηση που θα επιχειρήσουμε για την καταναμημένη εκτέλεση. Οι υπόλοιπες 90 που εκτελέστηκαν στο μικρότερο cluster έγιναν τόσο για την επαλήθευση των προηγούμενων παρατηρήσεων αλλά κυρίως για την μελέτη της κλιμακωσιμότητας που αναφέραμε. Εκτός των παραπάνω εκτελέσεων, για τους σκοπούς της συγκριτικής ανάλυσης που θέλουμε να κάνουμε με την κεντρική υλοποίηση του αλγορίθμου, εκτελέσαμε επιπρόσθετα και όλες τις περιπτώσεις με τα dataset που αναφέραμε στην ενότητα 4.1. Αυτές επιλέχθηκαν να πραγματοποιηθούν για το cluster με τους 9 nodes ενώ η σύγκριση αφορά όπως θα δούμε και στα επόμενα αποκλειστικά την χρονική επίδοση του αλγορίθμου για τις δύο αυτές περιπτώσεις.

Αναφορικά με το είδος των δεδομένων και τον τρόπο κατασκευής τους, ισχύουν και στην κατανεμημένη περίπτωση αυτά που αναφέραμε στην ενότητα 4.2 σχετικά. Χρησιμοποιήσαμε το ίδιο python script για να κατασκευάσουμε τα δεδομένα τα οποία δεν έχουν καμία προγενέστερη μορφή στο χώρο που να ευνοεί τη διαμόρφωση συγκεκριμένων συστάδων εκ προοιμίου. Με αυτόν τον τρόπο ορίζουμε, παρόμοια με την κεντρική υλοποίηση, ότι κάθε κατανεμημένη εκτέλεση στο Spark δεν θα συγκλίνει πριν εξαντλήσει τον μέγιστο αριθμό επαναλήψεων που θέτουμε μέσω του κώδικά μας.

Τέλος χρήσιμο είναι εδώ να αναφερθούμε στο μέγεθος της μνήμης που αφιερώνεται για χρήση cache στο spark όπως προκύπτει για τις επιλογές των cluster που κάναμε. Συγκεκριμένα για το cluster με τους 9 nodes το μέγεθος της συνολικής cache που είναι διαθέσιμο για κάθε εκτέλεση του αλγορίθμου υπολογίζεται από τον παρακάτω τύπο:

**Συνολική cache: 8 x 2600 x 0.9 x 0.6 =11.200 MB = 11.2 GB**

Στο παραπάνω τύπο 8 είναι ο συνολικός αριθμός των workers του συγκεκριμένου cluster. Το μέγεθος των 2600 MB προκύπτει από τα 3GB που δεσμεύονται σε κάθε κόμβο για την διεργασία worker του spark εκ των οποίων τα 2.6GB χρησιμοποιούνται από τον κάθε executor κατά τη διάρκεια της εκτέλεσης του αλγορίθμου. Το 1GB που αφαιρείται από το κάθε node αφορά στις ανάγκες σε μνήμη τόσο του λειτουργικού συστήματος αλλά και της διεργασίας datanode του hdfs που υπάρχει σε κάθε node, ενώ τα 400MB που υπολείπονται από τους executor είναι αυτά που χρησιμοποιεί ως κατώτατο όριο η διεργασία worker για τις δικές της ανάγκες. Ο πολλαπλασιασμός με τους αριθμούς 0.9 και 0.6 αφορά τα ποσοστά του heap που προορίζονται τελικώς για την cache και είναι αυτά που έχουμε αναφέρει στην ενότητα 2.2.3.4. Παρόμοιους υπολογισμούς μπορούμε να κάνουμε για να υπολογίσουμε την cache μνήμη για την περίπτωση του cluster με τους 5 nodes, καταλήγωντας στο ίδιο αποτέλεσμα.

## 5.2 Αλγόριθμος k-means στο Spark

Στην προσπάθεια ερμηνείας των πειραματικών αποτελεσμάτων είναι καίριας σημασίας η κατανόηση της πορείας εκτέλεσης που ακολουθεί το Spark για την εκτέλεση του αλγορίθμου, πράγμα το οποίο είναι εφικτό κυρίως λόγω του ότι το Spark είναι ένα project ανοικτού κώδικα και έτσι μπορέσαμε να μελετήσουμε την περίπτωση του k-means.

Στο Spark ορίζουμε αρχικά ως πηγή δεδομένων το hdfs αρχείο το οποίο είναι το dataset μας, ορίζουμε επίσης σε πόσα partitions το διαχωρίζουμε και διαλέγουμε επίσης τον τρόπο επιλογής των αρχικών κεντροειδών (centroids). Τα partitions είναι η οντότητα δεδομένων που αναλαμβάνει να επεξεργαστεί κάθε task που δρομολογείται σε κάθε node του cluster και είναι σημαντικό να διατηρούνται σε αριθμό κατ'ελάχιστο ίσο με το συνολικό αριθμό των cores που διαθέτει το cluster ώστε να διατηρείται η παράλληλη εκτέλεση του αλγορίθμου. Σε default τιμή το πλήθος των partition είναι ίσο με τα chunks του dataset που αποτελούν το hdfs αρχείο. Από τη στιγμή που ορίσουμε τα παραπάνω το κυρίως σώμα του αλγορίθμου περιλαμβάνει τα παρακάτω actions:

- **takeSamples:** Το action takeSamples είναι αυτό που εκτελείται πρώτο και είναι υπεύθυνο να διαλέξει τυχαία από όλο το dataset τα instances που θα ανακηρυχθούν αρχικά centroids. Προτού επιλεχθούν τα δείγματα αυτά, το action ενσωματώνει το φόρτωμα όλου του dataset ως HadoopRDD από το hdfs στη μνήμη του cluster καθώς και ένα count action το οποίο αναλαμβάνει να μετρήσει το πλήθος των instances για να διασφαλίσει ότι τα ζητούμενα clusters είναι μικρότερα από το συνολικό πλήθος των instances. Ουσιαστικά πρόκειται στο μεγαλύτερο μέρος του για ένα προπαρασκευαστικό στάδιο το οποίο όμως είναι και το πιο χρονοβόρο γιατί περιλαμβάνει το αρχικό διάβασμα όλου του dataset στην μνήμη (caching) και στο δίσκο και την αρχικοποίηση των κεντροειδών.
- **mapPartitions:** Το mapPartitions action είναι η υπολογιστική καρδιά του αλγορίθμου. Λαμβάνει κάθε partition του dataset ως μια επαναληπτική (iterator) δομή που αποτελείται από όλα τα instances που αυτό περιλαμβάνει, και εκτελεί σε κάθε ένα από αυτά μια συνάρτηση που δέχεται ως όρισμα. Η συνάρτηση αυτή στη περίπτωση του k-means είναι ορισμένη έτσι ώστε να υπολογίζει το κοντινότερο κεντροειδές για κάθε instance. Διατηρείται επίσης μια πινακοειδής δομή δεδομένων που έχει το μέγεθος του πλήθους των centroids και αποθηκεύεται εκεί για κάθε centroid, το πλήθος των instances που βρέθηκαν εγγύτερα στο συγκεκριμένο centroid καθώς και το άθροισμα των συντεταγμένων αυτών των instances. Η συνάρτηση λοιπόν αναλαμβάνει να ενημερώνει αυτή τη δομή και στο τέλος το mapPartitions την επιστρέφει ως ένα iterator αντικείμενο. Το αποτέλεσμα του κάθε mapPartitions αναλαμβάνεται να εξαχθεί από ένα task σε ένα κόμβο του cluster και αποθηκεύεται στη μνήμη και στο δίσκο αυτού του κόμβου. Η αποθήκευση αυτή των ενδιάμεσων αποτελεσμάτων ονομάζεται shuffle write.
- **reduceByKey:** Το mapPartitions action παράγει τόσα αποτελέσματα όσα και τα partition του dataset και αφορούν αποτελέσματα μόνο για το αντίστοιχο partition. Το reduceByKey action αναλαμβάνει να κάνει την συγχώνευση (merge) αυτών των αποτελεσμάτων ανά κλειδί (που στη περίπτωσή μας είναι το εκάστοτε centroid) σε μία ενιαία δομή. Αυτά είναι διάσπαρτα στους nodes του cluster και το reduceByKey λειτουργεί

σε δύο στάδια. Σε πρώτη φάση αναλαμβάνει να κάνει merge τοπικά σε κάθε κόμβο τα αποτελέσματα που έχουν προκύψει ως shuffle write. Σε δεύτερη φάση συγκεντρώνει (μέσω δικτύου) όλα τα αποτελέσματα του shuffle write ανά κλειδί σε ένα συγκεκριμένο κόμβο όπου γίνεται το τελικό merge που πλέον τα περιλαμβάνει όλους του υπολογισμούς συγκεντρωμένους ανά centroid.

- **collectAsMap:** Το collectAsMap είναι το action εκείνο με το οποίο τα ομαδοποιημένα ανά centroid αποτελέσματα του reduceByKey συγκεντρώνονται στον driver που εκτελείται στον master node και υπολογίζονται τα νέα centroids που θα χρησιμοποιηθούν στην επόμενη επανάληψη του αλγορίθμου. Επειδή ανα κεντροειδές έχει ήδη από τα προηγούμενα actions υπολογιστεί το άθροισμα των συντεταγμένων των instances που αντιστοιχούν σε αυτό καθώς και το πλήθος αυτών, ο υπολογισμός του νέου κεντροειδούς περιλαμβάνει μία απλή διαίρεση των ποσοτήτων αυτών.

Αυτό που σιωπηρά αποφύγαμε να επεξηγήσουμε στα προηγούμενα είναι ο τρόπος που τα centroids είναι γνωστά σε κάθε node του cluster ώστε να χρησιμοποιηθούν στους υπολογισμούς. Αυτό γίνεται με μία λειτουργία που παρέχει το spark και ονομάζεται broadcast variable. Το broadcast variable είναι μια read-only δομή που μπορεί να διαμοιραστεί σε όλους του nodes ενός cluster και να χρησιμοποιηθεί από τυχόν υπολογισμούς που την χρειάζονται καθόλη τη διάρκεια εκτέλεσης μιας εφαρμογής στο spark. Στη περίπτωσή μας ως broadcast variable ανακοινώνονται από τον driver στο τέλος κάθε επανάληψης τα νέα centroids και χρησιμοποιούνται από το mapPartitions action για τους υπολογισμούς των αποστάσεων.

Όσο αναφορά το λογικό πλάνο εκτέλεσης αυτό όπως έχουμε δει εισαγωγικά χωρίζεται σε jobs και stages. Στη περίπτωσή μας τα actions takeSamples και count ενοποιούνται σε ένα ξεχωριστό stage, το mapPartitions αποτελεί από μόνο του ένα stage ενώ τα reduceByKey και collectAsMap επίσης ενοποιούνται σε ένα ενιαίο stage, που παίρνει το όνομά του από το τελευταίο action. Για τα jobs έχουμε ένα που περιλαμβάνει το takeSamples stage και ένα που περιλαμβάνει τα υπόλοιπα δύο. Το τελευταίο job είναι αυτό που αποτελεί ουσιαστικά μία διακριτή επανάληψη του αλγορίθμου k-means στο spark. Όσο αναφορά το φυσικό πλάνο αυτό αποτελείται από τα task που δρομολογούνται στο cluster για κάθε stage που εκτελείται την εκάστοτε στιγμή, πάντα με σεβασμό το data locality που προσπαθεί να επιτύχει το spark.

## 5.3 Πειραματικά αποτελέσματα

Στις επόμενες υποενότητες θα παρουσιάσουμε τα αποτελέσματα και τις παρατηρήσεις των εκτελέσεών μας για το cluster με τους 9 nodes και για τα datasets του Πίνακα 5.2. Επικεντρωνόμαστε στις μετρήσεις που πραγματοποιήσαμε σχετικά με τη χρονική επίδοση των εκτελέσεων αυτών, το disk input και output που παρατηρήθηκε συνολικά από όλους του κόμβους του cluster καθώς και την δέσμευση της συνολικής μνήμης που χρειάστηκε για κάθε εκτέλεση.

### 5.3.1 Εξάρτηση από αριθμό επαναλήψεων

Για το πλήθος των επαναλήψεων του αλγορίθμου, όπως και στην κεντρική υλοποίηση επιλέξαμε την τιμή των 10 επαναλήψεων την οποία διατηρήσαμε σταθερή σε όλες τις εκτελέσεις που πραγματοποιήσαμε. Η παραπάνω επιλογή προέκυψε από την παρατήρησή μας ότι το πλήθος των επαναλήψεων επηρεάζει τα αποτελέσματα των μετρήσεών μας με ένα σταθερό παράγοντα αναλογίας ή και καθόλου.

Αναλυτικότερα όσο αναφορά το χρόνο εκτέλεσης παρατηρήσαμε σε δοκιμαστικές εκτελέσεις με διαφορετικές τιμές επαναλήψεων, ότι τα jobs που αφορούν το κυρίως σώμα του αλγορίθμου, δηλαδή τα stages mapPartitions και collectAsMap με πολύ μικρή απόκλιση έχουν ίδιο χρονικό κόστος. Το παραπάνω παρατηρήθηκε με την βοήθεια του spark web ui όπου διαθέτει χρονική μέτρηση για το κάθε job (και άρα για κάθε επανάληψη) και κρίναμε βάσει αυτού ότι κάθε εναλλαγή στον αριθμό των επαναλήψεων δεν προσφέρει αξία μελέτης.

Σχετικά με τη χρήση μνήμης παρατηρήσαμε ότι δεν υπάρχει εξάρτηση από τον αριθμό των επαναλήψεων. Αυτό είναι και το αναμενόμενο εφ'όσον μετά το πρώτο stage το σύνολο του dataset βρίσκεται cached στη μνήμη του cluster τουλάχιστον όσο αυτό είναι μικρότερο από τη συνολική μνήμη του cluster και από εκεί και πέρα κάθε επανάληψη δεν επιβαρύνει επιπρόσθετα την μνήμη. Αντιθέτως ενδιαφέρον παρουσιάζει η περίπτωση του disk input και του disk output η οποία επηρεάζεται από τον αριθμό των επαναλήψεων ιδιαίτερα στις περιπτώσεις που το dataset είναι μεγαλύτερο από την συνολική μνήμη του cluster άρα το Spark χρησιμοποιεί και το δίσκο για να αποθηκεύσει partitions και ενδιάμεσα αποτελέσματα. Όμως ακόμα και σε αυτή τη περίπτωση παρατηρείται τελικώς ότι το μέγεθος το οποίο διακινείται από και προς το δίσκο σε αυτές τις περιπτώσεις αφορά μια σχεδόν σταθερή τιμή ανά επανάληψη και τελικώς και αυτή η μετρική αναμένεται να αυξάνεται κατά ένα σταθερό παράγοντα για κάθε επόμενη επανάληψη.

### 5.3.2 Μέτρηση CPU

Η ανάλυση των μετρικών που αφορούν την χρήση της cpu στην κατανεμημένη εκτέλεση είναι δύσκολη να γίνει λόγω των ακαθόριστων τιμών που λαμβάνουν αυτές κατά τη διάρκεια εκτέλεσης για κάθε node του cluster. Συγκεκριμένα λόγω της πληθώρας των λόγων που μπορούν να πυροδοτήσουν διαφορετικά γεγονότα για τη cpu (π.χ garbage collection, διάβασμα από το hdfs) και λόγω της μη ύπαρξης χρονικής ομοιογένειας ως προς την εμφάνιση τους για όλους του nodes του cluster η αναλυτική μελέτη των μετρικών αυτών δεν μπορεί να οδηγήσει σε εξαγωγή ασφαλών συμπερασμάτων.

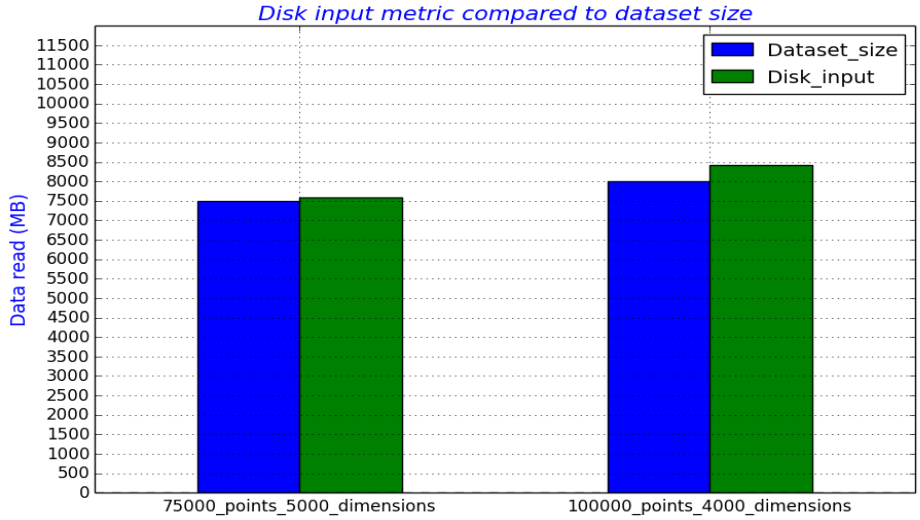
Παρόλα αυτά η γενική εικόνα που αποκτήσαμε μελετώντας τις μετρικές για κάθε node είναι μια μικρή αύξηση του cpu_wio στα αρχικά στάδια της εκτέλεσης καθώς τότε είναι που κάθε κόμβος διαβάζει τα αντίστοιχα partitions από το hdfs. Στη συνέχεια της εκτέλεσης και καθώς εκτελούνται οι κύριες υπολογιστικές διαδικασίες του αλγορίθμου ανά partition, παρατηρήσαμε τη φυσιολογική αύξηση του cpu_user. Καθώς το cpu_user εκφράζεται ως ποσοστό του χρόνου που το σύνολο των cores εκτελεί κώδικα user space οι τιμές αυτές αγγίζουν οριακά τις τιμές 90%-95% σε αρκετές στιγμές της εκτέλεσης υποδηλώνοντας την ταυτόχρονη εκτέλεση των task που αναθέτονται σε κάθε node. Όσο αναφορά την μέτρηση

cpu_system αυτή παρατηρήθηκε σε πολύ μικρές τιμές, πράγμα που υποδεικνύει ότι στη κατανεμημένη εκτέλεση δεν υπάρχει σημαντικός χρόνος που να καταναλίσκεται στην εξυπηρέτηση system calls. Η μόνη διαφοροποίηση στη παραπάνω περιγραφή παρατηρείται για τον master node που η συμμετοχή τους στους υπολογισμούς περιορίζεται μόνο στο τέλος κάθε επανάληψης όπου χρειάζεται να εκτελέσει κώδικα για να υπολογίσει τα νέα centroids. Ειδικά για τον master παρατηρούμε σημαντική άυξηση του cpu_user μόνο για τις συγκεκριμένες χρονικές στιγμές.

### 5.3.3 Disk input

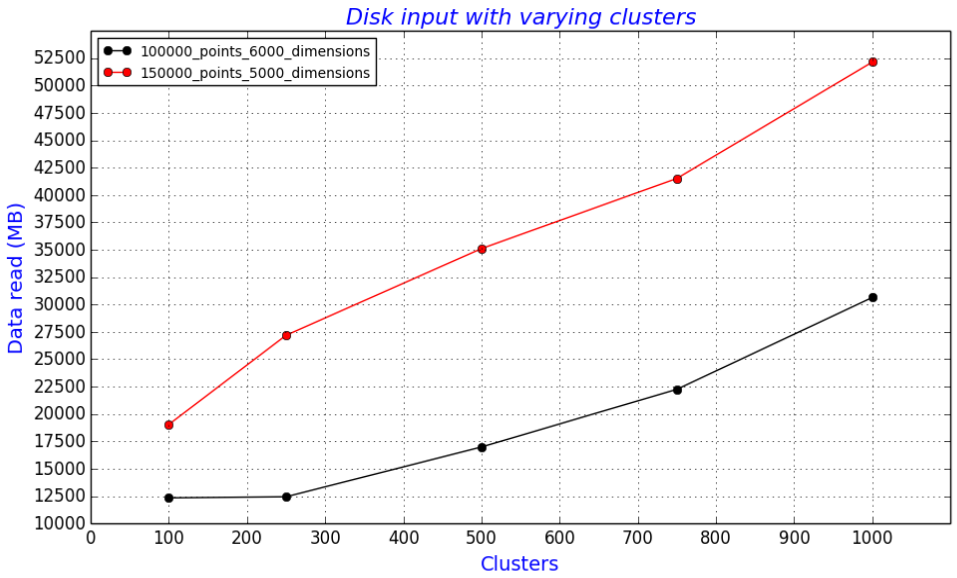
Το disk input που μετρήσαμε αφορά το σύνολο των δεδομένων τα οποία αναγνώστηκαν από το δίσκο κατά τη διάρκεια της εκτέλεσης του αλγορίθμου και αποτελεί αθροιστική μέτρηση όλων των επιμέρους μετρήσεων για κάθε node του cluster. Η μετρούμενη τιμή ενσωματώνει αρχικά το μέγεθος του ίδιου του dataset που χρησιμοποιούμε στην εκτέλεση αφού αυτό βρίσκεται αποθηκευμένο στο hdfs και άρα στους τοπικούς δίσκους των nodes του cluster. Επίσης ενσωματώνει και την τιμή εκείνη που αφορά στο διάβασμα δεδομένων κατά τη διάρκεια της εκτέλεσης ως απόρροια της χρήσης του δίσκου ως δευτερεύοντος μέσου αποθήκευσης. Καθώς το spark πάντα εξαντλεί τα αποθέματα της cache που διαθέτει το cluster για λόγους ταχύτητας πρόσβασης, η τελευταία αυτή τιμή προκύπτει όταν έχουμε ένα dataset το οποίο είναι μεγαλύτερο από το μέγεθος της διαθέσιμης cache και άρα το spark χρησιμοποιεί τον δίσκο ως βοηθητικό μέσο αποθήκευσης και δεδομένων αλλά και ενδιάμεσων αποτελεσμάτων.Από τα παραπάνω ήδη φαίνεται ότι αναμένουμε δύο διαφορετικές συμπεριφορές σε σχέση με το μετρούμενο disk input που εξαρτώνται από το μέγεθος του dataset συγκριτικά με το μέγεθος της διαθέσιμης cache όπως αυτή υπολογίστηκε στην ενότητα 5.1 (11.2 GB).

Η πρώτη περίπτωση που αφορά τα dataset εκείνα τα οποία υπολείπονται σε μέγεθος της συνολικής cache του cluster φαίνεται γραφικά στο Σχήμα 5.1 παρακάτω. Σε αυτό βλέπουμε τα μεγέθη των dataset για δύο διαφορετικές περιπτώσεις εκτελέσεων σε σύγκριση με τις μετρήσεις του disk input. Η μέτρηση του disk input υπολογίστηκε ως ο μέσος όρος των μετρήσεων που πήραμε για κάθε εκτέλεση αυτών των dataset με διαφορετική τιμή clusters. Παρατηρούμε λοιπόν ότι όσο οι τιμές του πλήθους των instances και των dimensions περιγράφουν ένα dataset μικρότερης χωρητικότητας από αυτής της cache του cluster η τιμή του disk input που μετριέται συνολικά στο cluster εκφράζει μόνο τη διαδικασία ανάγνωσης του dataset από τους τοπικούς δίσκους των κόμβων του cluster. Εδώ διαφαίνεται και η λειτουργία του caching όπου καθόλη τη διάρκεια της εκτέλεσης του αλγορίθμου το Spark έχει διαθέσιμα όλα τα δεδομένα του στην κύρια μνήμη των nodes και δεν επιβαρύνεται από χρονοβόρες αναγνώσεις από το δίσκο.




*Σχήμα 5.1: Μέτρηση disk input για datasets μικρότερου μεγέθους από την cache*

Μεγαλύτερο ενδιαφέρον παρουσιάζουν οι περιπτώσεις dataset που υπερβαίνουν σε μέγεθος την cache. Δύο τέτοιες περιπτώσεις dataset με τις αντίστοιχες τιμές disk input για μεταβαλλόμενο αριθμό clusters φαίνονται γραφικά στο Σχήμα 5.2 παρακάτω. Η περίπτωση των 100000 instances και 6000 dimensions πρόκειται για dataset μεγέθους 12GB ενώ η περίπτωση των 150000 instances και 5000 dimensions είναι μεγέθους 15GB.




*Σχήμα 5.2: Μέτρηση disk input για datasets μεγαλύτερου μεγέθους από την cache*

Από το Σχήμα 5.2 είναι φανερή η πολύ σημαντική επίδραση της μεταβολής του disk input καθώς αυξάνουμε το πλήθος των cluster για ένα συγκεκριμένο dataset. Αυτή εξηγείται από το διάβασμα των αποτελεσμάτων του shuffle write που γίνεται με το πέρας κάθε επανάληψης του αλγορίθμου από το stage reduceByKey και λόγω ανεπάρκειας κύριας μνήμης έχουν αποθηκευτεί στους τοπικούς δίσκους των nodes. Ακόμα όμως πιο σημαντική

επίδραση στην μέτρηση του disk input είναι όπως φαίνεται η αύξηση του πλήθους των instances συγκρίνοντας τις τιμές ανάμεσα στις δύο παραπάνω καμπύλες. Αυτή μεταφράζεται ως το αποτέλεσμα της αναγκαστικής αποθήκευσης των partitions του dataset στο δίσκο κατά τη διάρκεια της εκτέλεσης.

Για την παρουσίαση αποτελεσμάτων για διαφορετικές τιμές dimensions και πλήθους instances για περιπτώσεις dataset με μεγαλύτερο μέγεθος από την cache, περιοριστήκαμε από τις ικανότητες του spark να διαχειρισθεί αποτελεσματικά τέτοια datasets. Όπως αναφέραμε και εισαγωγικά οποιαδήποτε αύξηση του πλήθους των instances και των dimensions πέρα από τις τιμές 150000 και 6000 αντίστοιχα αδυνατούσε να εκτελεστεί ομαλά κυρίως λόγω προβλημάτων με το περιθώριο μνήμης ανα node που είχαμε ορίσει στο cluster μας. Το παραπάνω γεγονός μας περιόρισε στη ανάκτηση ικανού πλήθους δειγμάτων για τη μελέτη μας. Θεωρητικώς το μοντέλο εκτέλεσης που προτείνει το spark δεν περιορίζεται από το μέγεθος των dataset σε σχέση με τους διαθέσιμους πόρους του cluster αφού μπορεί πάντα να θυσιάσει την αποδοτικότητα με σκοπό την ομαλή ολοκλήρωση των jobs. Η παρατήρησή μας αυτή ενδεχομένως να αποκαλύπτει κάποιες αδυναμίες του spark στο κομμάτι της διαχείρισης μνήμης, τουλάχιστον στα πλαίσια της εκτέλεσης του αλγορίθμου k-means που μελετούμε.

### 5.3.4 Disk output

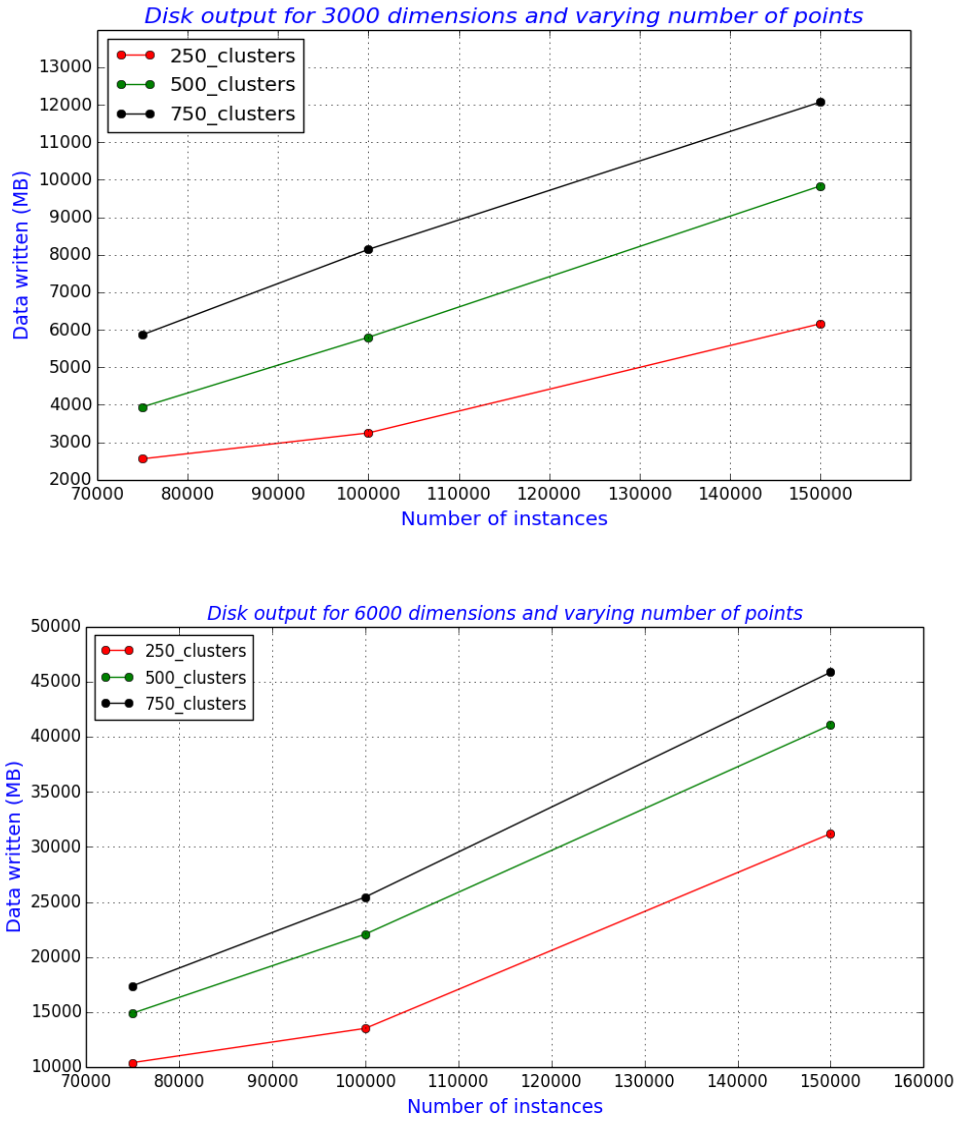
Η μέτρηση του disk output έχει δύο σημαντικούς παράγοντες που την επηρεάζουν. Ο πρώτος αφορά το shuffle write που γίνεται στο τέλος του stage mapPartitions και ένα μεγάλο ποσοστό αυτού αποθηκεύεται στο δίσκο αλλά και στην ανάγκη εγγραφής partitions τα οποία δεν χωρούν στην κύρια μνήμη. Ο παράγοντας του shuffle write είναι εντονότερος για εκείνα τα datasets τα οποία είναι μικρότερα σε μέγεθος της συνολικής cache ενώ η επίδραση του στη μέτρηση υπερκαλύπτεται όσο τα dataset γίνονται μεγαλύτερα και η μετακίνηση partition στο δίσκο γίνεται εντονότερη.

Στα επόμενα θα αναλύσουμε σε τρεις επιμέρους υποενότητες τις μετρήσεις που λάβαμε για το disk output συναρτήσει των μεταβολών του πλήθους των instances, των dimensions αλλά και των clusters. Σκοπός μας θα είναι να δούμε πως κάθε μία παράμετρος ξεχωριστά επηρεάζει τη συγκεκριμένη μέτρηση αλλά και η εξαγωγή ενός γενικότερου συμπεράσματος με βάση αυτές τις παρατηρήσεις.

#### 5.3.4.1 Επίδραση πλήθους instances

Η μεταβολή του disk output για μεταβαλλόμενες τιμές του πλήθους των instances φαίνεται γραφικά στο Σχήμα 5.3 για δύο διαφορετικές τιμές dimensions και για διαφορετικές τιμές clusters. Σε αυτό η περίπτωση των 3000 dimensions αφορά dataset που για όλες τις τιμές του πλήθους των instances έχει μέγεθος μικρότερο της συνολικής cache του cluster ενώ για αυτήν των 6000 dimensions η περίπτωση με τα 150000 αφορά dataset που ξεπερνάει σε μέγεθος την cache.

Αυτό που παρατηρήσαμε είναι υπάρχει μία εξομαλυμένη αυξητική τάση των δεδομένων που εγγράφονται στο δίσκο καθώς αυξάνουμε το πλήθος των instances με γραμμική συμπεριφορά για τις περιπτώσεις dataset μικρότερου μεγέθους από τη cache ανεξαρτήτως του πλήθους των cluster. Στις τελευταίες, από τη σύγκριση που κάναμε με τις τιμές του shuffle write που μας παρείχε το spark web ui είδαμε ότι βρίσκονται σχεδόν σε ισορροπία οι μετρήσεις. Λέμε σχεδόν, γιατί όπως αναφέραμε και στην εισαγωγική ενότητα σχετικά με το caching του spark υπάρχει κομμάτι του διαθέσιμου heap κάθε executor το οποίο εξυπηρετεί ένα ορισμένο ποσοστό ανάγκης σε μνήμη ειδικά για τους σκοπούς του shuffle write, άρα αυτό το ποσοστό δεν καταγράφεται στις μετρήσεις μας.



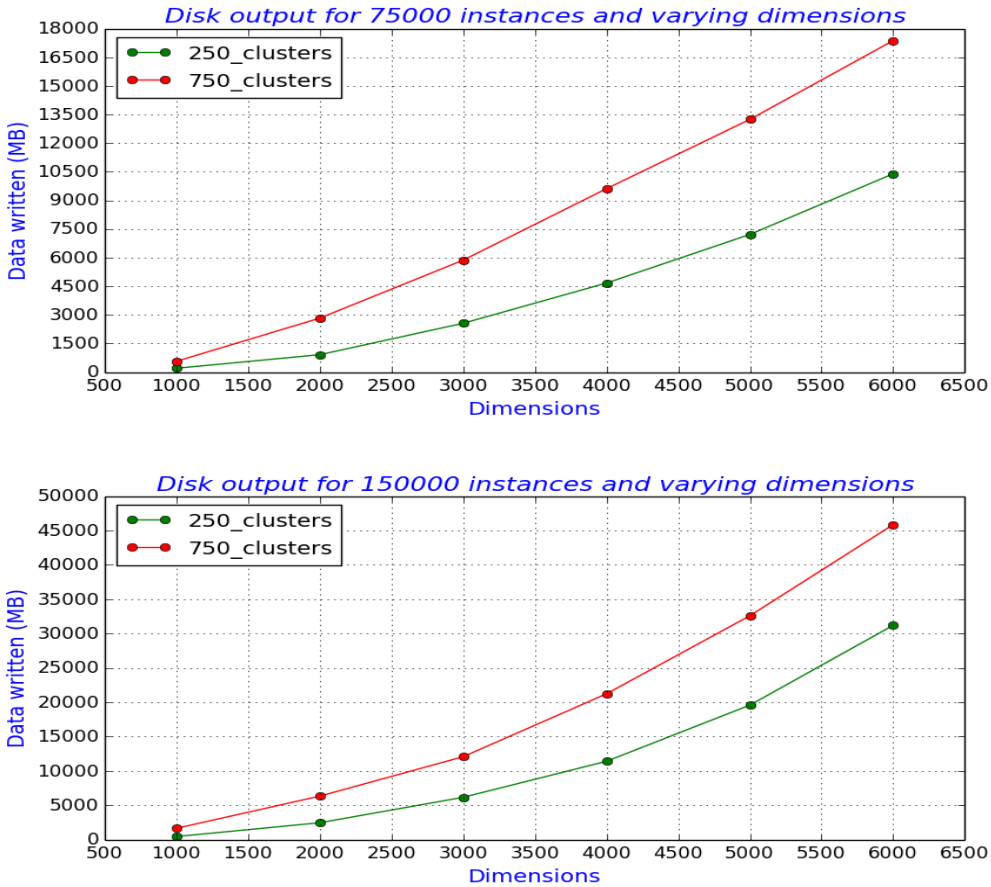

*Σχήμα 5.3: Disk output για διαφορετικές τιμές πλήθους instances*

Η γραμμική αυτή συμπεριφορά "διασπάται" όταν πλέον έχουμε να διαχειριστούμε datasets μεγαλύτερου μεγέθους. Αυτό παρατηρείται στις γραφικές για τα 6000 dimensions του Σχήματος 5.3 καθώς αυξάνουμε την τιμή των instances από 100000 σε 150000. Εκεί

παρατηρούμε μία απότομη αύξηση του disk output η οποία ερμηνεύεται ως χρήση του δίσκου για την αποθήκευση και partitions κατά τη διάρκεια εκτέλεσης.

### 5.3.4.2 Επίδραση πλήθους dimensions

Η παράμετρος των dimensions επηρεάζει σημαντικά το μέγεθος των δεδομένων shuffle write της κατανεμημένης εκτέλεσης. Αυτό είναι εύκολα αντιληπτό αν ανατρέξουμε στη περιγραφή μας για την πινακοειδή μορφή που έχουν τα αποτελέσματα που εξάγει το mapPartitions stage. Η παρατήρησή μας αυτή επιβεβαιώνεται και από τα αποτελέσματα των μετρήσεών μας που φαίνονται στο Σχήμα 5.4 για δύο διαφορετικές περιπτώσεις cluster και δυο διαφορετικές τιμές πλήθους instances.



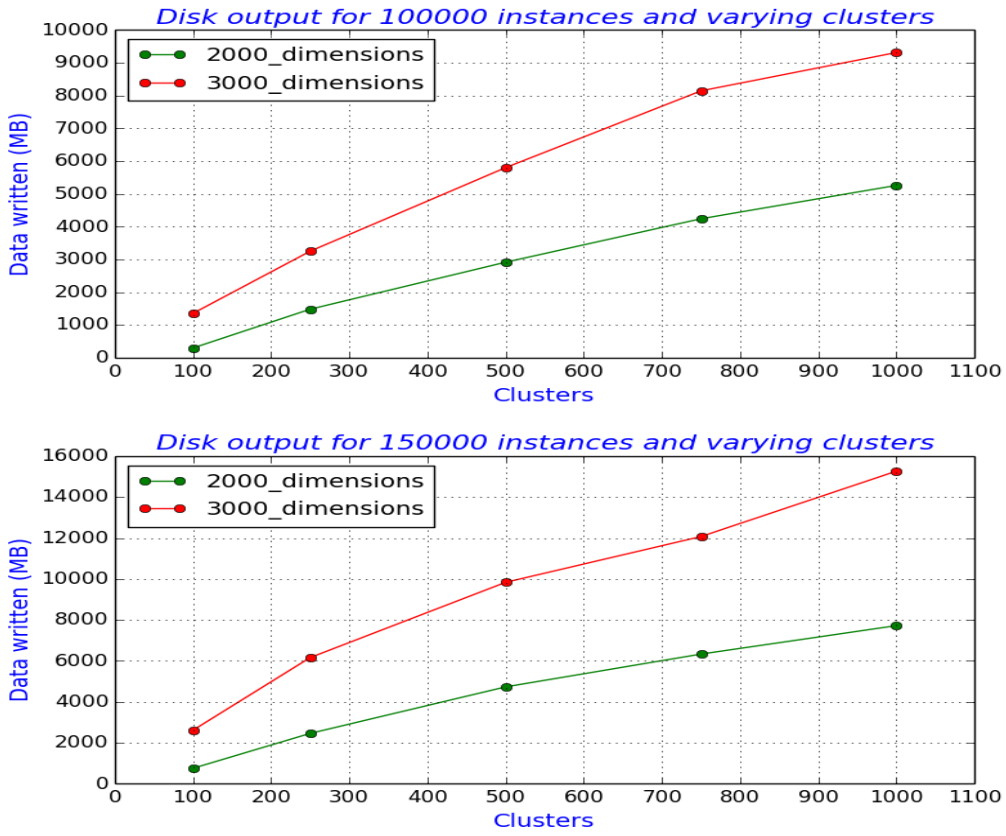

*Σχήμα 5.4: Disk output για διαφορετικές τιμές dimensions*

Στο παραπάνω σχήμα παρατηρούμε μία ενιαία αυξητική συμπεριφορά ανεξαρτήτως του πλήθους των cluster που επιλέγουμε. Το ενδιαφέρον είναι ότι αυτή η αύξηση σε όλες τις περιπτώσεις θα μπορούσε να χαρακτηριστεί ως παραβολικού τύπου πράγμα που σημαίνει ότι δεν έχουμε ομαλή αύξηση της μετρούμενης τιμής του disk output καθώς μεταβάλλουμε το

πλήθος των dimensions. Αυτή η παραβολικού τύπου συμπεριφορά γίνεται ακόμα εντονότερη όταν μιλάμε για dataset που έχουν μεγαλύτερο μέγεθος από τη cache του cluster όπως αυτό συμβαίνει για τις καμπύλες του δεύτερου γραφήματος που αφορά τα 150000 instances. Αυτό εξηγείται από το γεγονός ότι σε αυτές, τα dimensions επηρεάζουν ταυτόχρονα τόσο το μέγεθος των partitions που μπορούν να αποθηκευτούν στη cache αλλά και το μέγεθος των shuffle write δεδομένων που παράγονται ενδιάμεσα στην εκτέλεση.

### 5.3.4.3 Επίδραση πλήθους cluster

Η παράμετρος των cluster που ζητάμε να γίνει η συσταδοποίηση είναι μαζί με τα dimensions αυτή που επηρεάζει το μέγεθος των shuffle write δεδομένων λόγω της μορφής που έχουμε αναφέρει ότι αυτά έχουν. Έτσι είναι λογικό να έχουμε μία αυξανόμενη τάση για χρήση του δίσκου όσο η τιμή των cluster αυξάνεται. Αυτό διαφαίνεται από τις γραφικές του Σχήματος 5.5 παρακάτω για διαφορετικές τιμές πλήθους instances και dimensions.




*Σχήμα 5.5: Disk output για διαφορετικές τιμές cluster*

Εκείνο που παρατηρείται από το παραπάνω γράφημα είναι η σχετικά ομαλή γραμμική αύξηση του disk output με την αύξηση των cluster. Επειδή όμως κάθε καμπύλη εκφράζει συγκεκριμένο dataset με ορισμένο πλήθος partitions αυτό σημαίνει ότι η αύξηση της τιμής

του disk output στα παραπάνω γραφήματα εκφράζει αποκλειστικά το μέγεθος του shuffle write της εκτέλεσης για όλες τις επαναλήψεις σε σχέση με το πλήθος των cluster. Έτσι μπορούμε να συμπεράνουμε ότι η τιμή των cluster έχει μικρότερη συμβολή στην αύξηση του disk output από ότι τα dimensions που εξετάσαμε προγενέστερα. Το τελευταίο συμπέρασμα σχετίζεται βέβαια και με τις επιλεγμένες τιμές για τις δύο παραμέτρους όπου έχουμε επιλέξει τα dimensions να παίρνουν πολύ μεγαλύτερες συγκριτικά με το πλήθος των cluster.

Κλείνοντας με αυτή την υποενότητα τη μελέτη μας σχετικά με την μέτρηση του disk output, κρίνουμε σκόπιμο να κάνουμε μία μικρή περίληψη των παρατηρήσεών μας. Αυτές συνοψίζονται στο γεγονός ότι η μετρική του disk output εξαρτάται κυρίως από το μέγεθος του dataset συγκρινόμενο με την μνήμη cache του cluster. Άμα αυτό είναι μικρότερο τότε αυτό διατηρείται σε μικρές τιμές που αντιστοιχούν σχεδόν εξ'ολοκλήρου στο shuffle write και άρα επηρεάζονται κυρίως από την εναλλαγή του πλήθους των dimensions και των cluster. Στις περιπτώσεις που αυτό είναι μεγαλύτερο τότε κυρίαρχοι παράγοντες που καθορίζουν την τιμή του είναι πλήθος των instances και τα dimensions γιατί αυτές οι δύο παράμετροι ουσιαστικά ορίζουν τον αριθμό των partitions που θα πρέπει να διατηρούνται στο δίσκο και να διαβάζονται και να εγγράφονται εκεί. Στην τελευταία αυτή περίπτωση είναι σαφές ότι η παρατηρούμενες τιμές είναι κατά πολύ μεγαλύτερες.

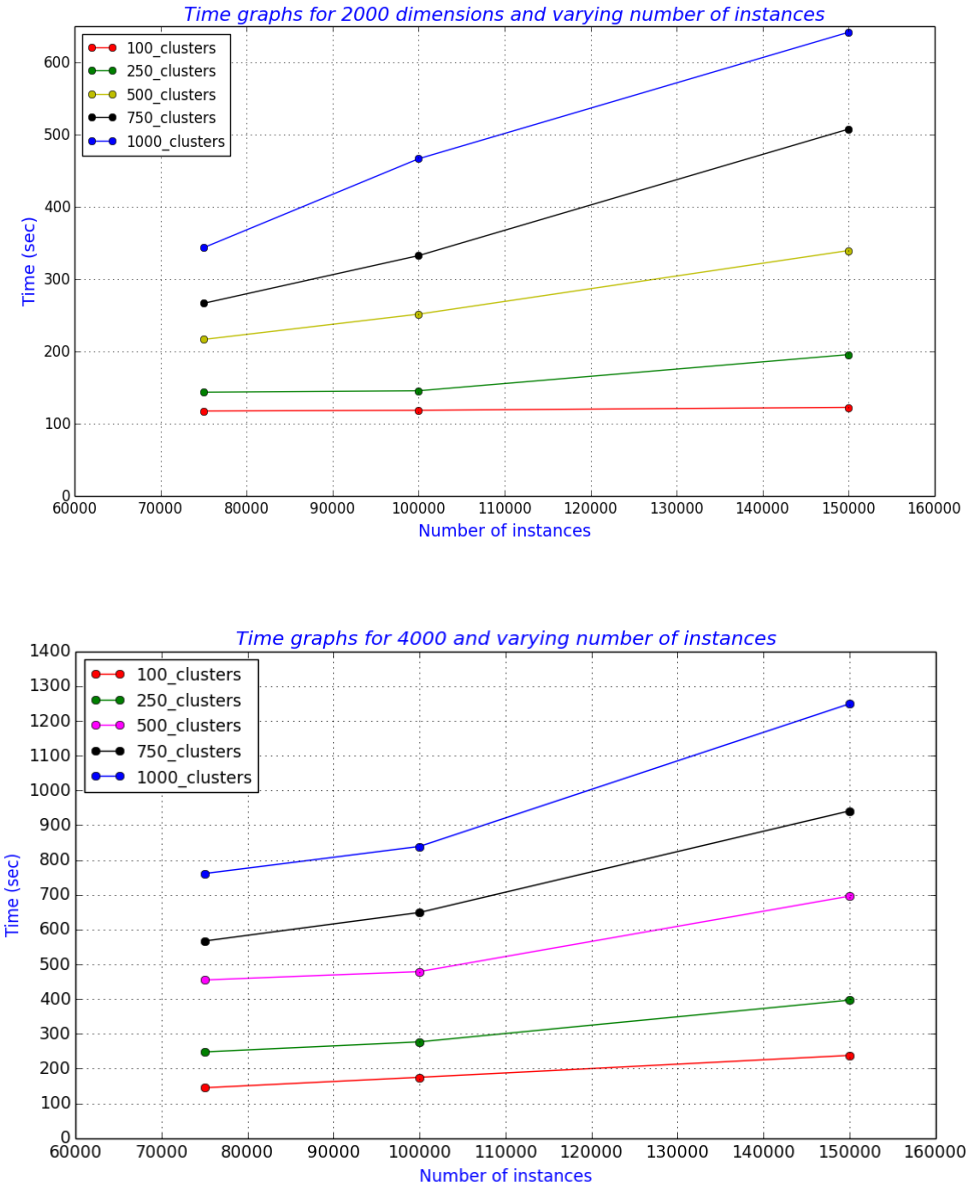
### 5.3.5 Χρόνος εκτέλεσης

Η μέτρηση του χρόνου εκτέλεσης στη κατανεμημένη υλοποίηση του spark είναι μια απλή διαδικασία όσο αφορά την μέτρησή του αλλά έχει αυξημένη πολυπλοκότητα όσο αφορά την ανάλυσή του. Αυτό συμβαίνει διότι υπάρχει ένας μεγάλος αριθμός ετερογενών παραγόντων που την επηρεάζουν, και εκτείνεται επί παραδείγματι από τον χρόνο που απαιτείται να δρομολογηθούν τα tasks από τον scheduler του spark στους executors μέχρι το χρόνο που δαπανάται σε garbage collection σε ένα task και μπορεί να είναι μερικές φορές υπολογίσιμος.

Στις επόμενες ενότητες, σύμφωνα με την τακτική παρουσίασης που έχουμε ακολουθήσει έως τώρα, θα παρουσιάσουμε τους χρόνους εκτέλεσης που μετρήσαμε και πως αυτοί επηρεάζονται καθώς μεταβάλλουμε μία παράμετρο εκτέλεσης κάθε φορά. Θα προσπαθήσουμε να αποδομήσουμε το συνολικό χρονικό κόστος στους επιμέρους παράγοντες που το αποτελούν ενώ μας ενδιαφέρει να δούμε ποια από τις παραμέτρους επηρεάζει αποφασιστικότερα τις μετρήσεις μας.

#### 5.3.5.1 Επίδραση πλήθους instances

Σχετικά με την επίδραση του πλήθους των instances στη χρονική εκτέλεση, δείχνουμε στο Σχήμα 5.6 τις περιπτώσεις για δύο διαφορετικά dimensions και για όλες τις διαφορετικές εκτελέσεις για τις τιμές των cluster.




*Σχήμα 5.6: Χρόνος εκτέλεσης για διαφορετικές τιμές πλήθους instances*

Αυτό που παρατηρούμε στο Σχήμα 5.6 είναι ότι η αύξηση του πλήθους των instances δεν έχει σημαντικό χρονικό αντίκτυπο καθώς αυξάνουμε την τιμή του σε αρκετές περιπτώσεις. Ειδικότερα για μικρές τιμές του cluster θα μπορούσαμε να πούμε ότι η μέτρησή μας αυξάνεται με πολύ αργό ρυθμό σε σχέση με μια σημαντική αύξηση στο πλήθος των instances. Για μεγαλύτερες τιμές του cluster υπάρχει πιο αισθητή διαφορά στο μετρούμενο χρόνο όπως αυτό φαίνεται από τις καμπύλες για τα 1000 clusters και για τις δύο περιπτώσεις παραπάνω.

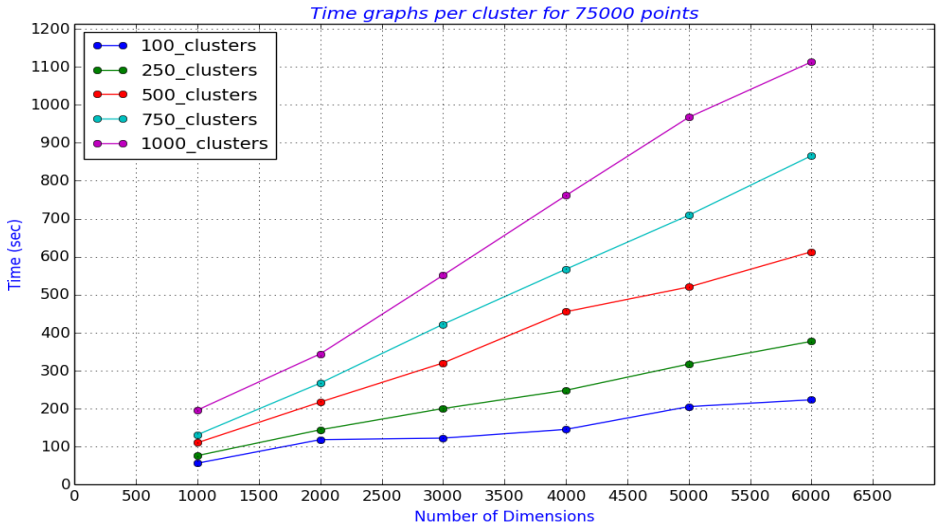
Αύξηση του πλήθους των instances συνεπάγεται αύξηση του πλήθους των partitions που πρέπει να δημιουργηθούν από το dataset και ισόποση αύξηση στο πλήθος των task που πρέπει να δρομολογηθούν στο cluster για την ολοκλήρωση της εκτέλεσης. Δεδομένου επίσης του γεγονότος ότι για τις εκτελέσεις με ίδιο αριθμό dimensions και cluster παρατηρήσαμε από το spark web ui ίδιους χρόνους εκτέλεσης, η παραπάνω μικρή αύξηση την χρονική

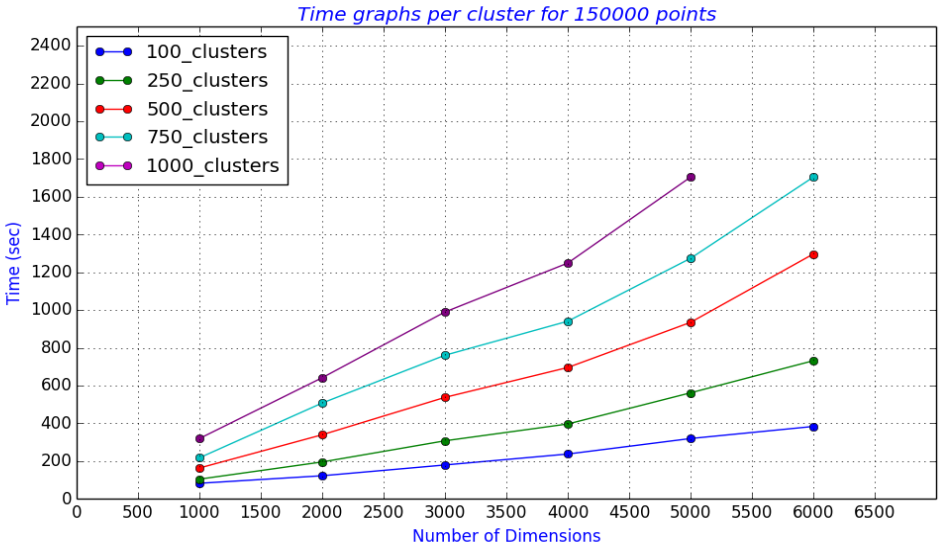
εκτέλεσης μπορεί να αποδοθεί στο χρόνο που καταναλώνεται να διαβαστούν τα επιπρόθεστα partitions από το hdfs στην φάση του takeSamples stage και στους χρόνους που καταναλώνει το spark στο scheduling των tasks και την ικανοποίηση του data locality.

Από τα παραπάνω φαίνεται ότι το spark κάνει πολύ καλή δουλειά και στους δύο τομείς. Διαθέτει αρχικά πολύ ικανό scheduler που πετυχαίνει πολύ καλούς χρόνους όσο αφορά την ανάθεση task σε κόμβους του cluster αλλά και στο serialization και deserialization που χρειάζονται αυτά ώστε να μεταδίδονται μέσω δικτύου. Επίσης συνεργάζεται πολύ καλά με το api του hdfs και πετυχαίνει καλό data locality με την έννοια ότι δρομολογεί tasks στους datanodes που έχουν τοπικά το chunk το οποίο θα επεξεγαστεί ο τοπικός executor. Επίσης αποφασιστικό ρόλο παίζει και το caching, καθώς στην περιπτώσεις που εκθέτουμε στις γραφικές μας έχουμε το σύνολο των δεδομένων cached στην μνήμη του cluster καθόλη τη διάρκεια εκτέλεσης του αλγορίθμου.

### 5.3.5.2 Επίδραση πλήθους dimensions

Η εναλλαγή του πλήθους των dimensions επιβαρύνει με δύο τρόπους την χρονική εκτέλεση του αλγορίθμου. Από την μία διαμορφώνει μαζί με το πλήθος των instances που είδαμε προηγουμένως, το μέγεθος του dataset άρα και τον συνολικό αριθμό των tasks που πρέπει να δρομολογηθούν. Εκτός όμως από αυτό αποτελεί και παράγοντα χρονικού κόστους στον υπολογισμό των αποστάσεων κάθε instance από κάθε κεντροειδές σε κάθε επανάληψη. Ο τελευταίος αποτυπώνεται ως αύξηση του παρατηρούμενου χρόνου εκτέλεσης για κάθε ένα task που εκτελείται επί ενός partition στο cluster. Στο Σχήμα 5.7 βλέπουμε γραφικά 2 περιπτώσεις για διαφορετικό πλήθος instances και για όλες τις διαφορετικές τιμές cluster.






*Σχήμα 5.7: Χρόνος εκτέλεσης για διαφορετικές τιμές dimensions*

Σχετικά με την τελευταία μέτρηση στη δεύτερη γραφική για τα 150000 instances, 6000 dimensions και 1000 cluster εκλείπει γιατί όπως έχουμε ήδη σχολιάσει αδυνατούσε να εκτελεστεί λόγω προβλημάτων ανεπάρκειας μνήμης του spark. Αυτό που παρατηρούμε ανεξαρτήτως πλήθους instances αλλά και πλήθους cluster είναι μια γραμμική εξάρτηση του χρόνου από το πλήθος των dimensions. Ο συντελεστής αναλογίας της γραμμικής αυτής εξάρτησης αυξάνεται καθώς αυξάνεται και το πλήθος των cluster. Αυτό συμβαίνει διότι οι παράμετροι του dimension και του cluster συνεργατικά επιβαρύνουν το κόστος υπολογισμού της ευκλείδιας απόστασης και της αναζήτησης του εγγύτερου centroid για κάθε instance στο mapPartitions stage. Το τελευταίο επιβεβαιώθηκε με την παρατήρηση από το spark web ui της αύξησης του χρόνου εκτέλεσης των μεμονομένων task για τις διαφορετικές εκτελέσεις. Αυτή η χρονική επιβάρυνση λόγω την σημασίας της, είναι και ένα σημείο της εκτέλεσης του αλγορίθμου που δίνεται βαρύτητα από τις εκάστοτε υλοποιήσεις σχετικά με τον τρόπο που μπορεί να βελτιστοποιηθεί. Συγκεκριμένα το spark χρησιμοποιεί στο κώδικά του ειδικά πακέτα λογισμικού (breeze) που περιέχουν βιβλιοθήκες (netlib-java,jblas) που σχετίζονται με πράξεις διανυσμάτων και πινάκων και στη βάση τους χρησιμοποιούν native Fortran ρουτίνες για την επιτάχυνση αυτών των πράξεων όσο είναι δυνατό.

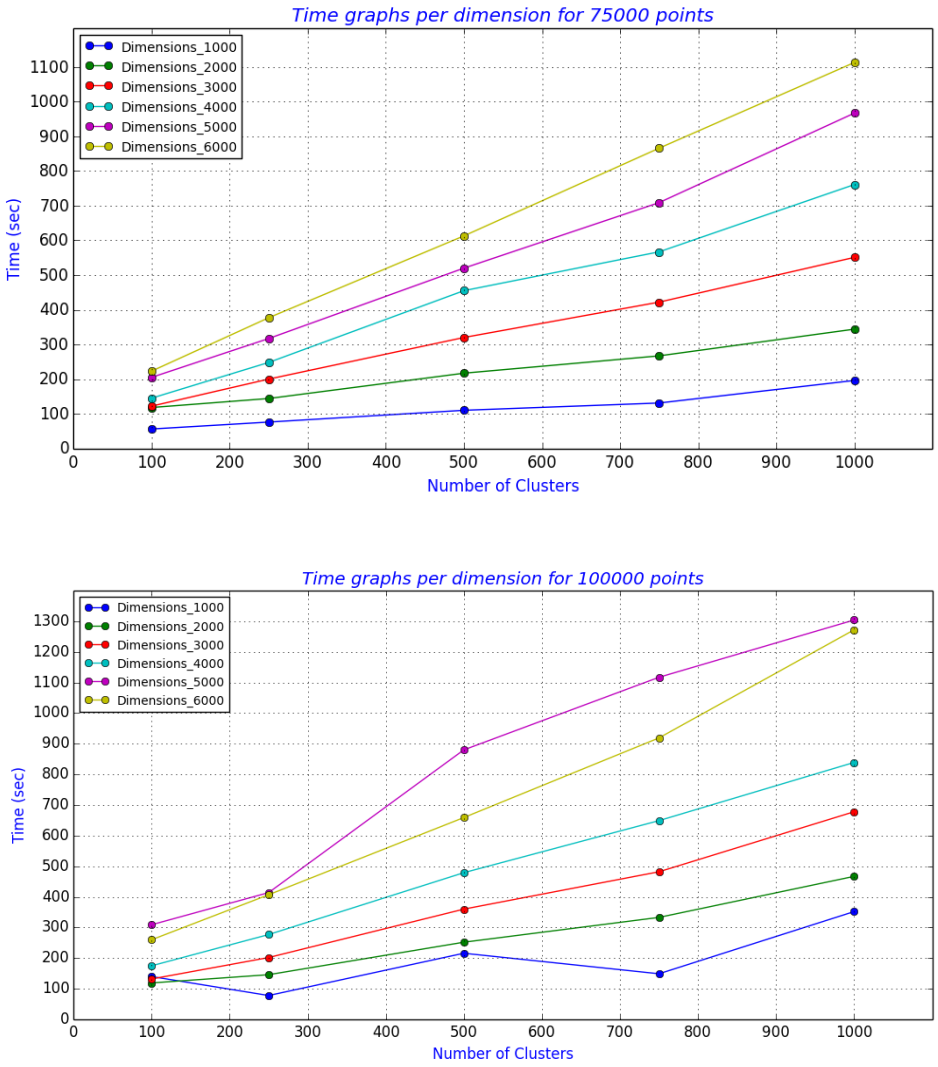
Ακόμα ένα ένας παράγοντας που επηρεάζει τη χρονική εκτέλεση καθώς αυξάνουμε τις τιμές των dimensions σχετίζεται με τις παρατηρήσεις που κάναμε για το disk output και input που είδαμε σε προηγούμενες ενότητες. Εκεί είδαμε εξάρτηση των παραπάνω δύο μετρικών με την αύξηση των dimensions. Αυτό μεταφράζεται άμεσα σε χρόνο που αναλώνεται σε αιτήσεις ανάγνωσεις και εγγραφής στο δίσκο για τους λόγους που ήδη έχουμε περιγράψει. Αυτός ο χρόνος έχει προφανώς αρνητική συμβολή στη χρονική επίδοση του συστήματος αφορά τις μετρήσεις που βλέπουμε στο Σχήμα 5.7.

Τέλος τα dimensions παίζουν μεγάλο ρόλο στο μέγεθος (μαζί με τα cluster) των ενδιάμεσων αποτελεσμάτων shuffle write που προκύπτουν κατά τη διάρκεια εκτέλεσης. Αυτά είναι δεδομένα που πρέπει με το πέρας κάθε επανάληψης να διαβαστούν από nodes του cluster όπου γίνεται η συγκέντρωση όλων των τιμών ανά centroid. Η μετακίνηση αυτών των

δεδομένων μέσω δικτύου είναι επίσης ένας παράγοντας που επηρεάζει χρονικά την εκτέλεση του αλγορίθμου και πρέπει να λαμβάνεται υπόψη.

#### 5.3.5.3 Επίδραση πλήθους cluster

Για την επίδραση του πλήθους των cluster παραθέτουμε γραφικά τα αποτελέσματά μας για δύο διαφορετικές τιμές instances και για όλες τις διαφορετικές τιμές που επιλέξαμε για τα dimensions στις εκτελέσεις μας στο Σχήμα 5.8.




*Σχήμα 5.8: Χρόνος εκτέλεσης για διαφορετικές τιμές cluster*

Το πλήθος των cluster της κάθε εκτέλεσης δεν επηρεάζει το μέγεθος του dataset άρα δεν έχει κανένα αντίκτυπο στο πλήθος των task που θα εκτελεστούν. Η μοναδική χρονική επιβάρυνση που προσθέτει αφορά αποκλειστικά τους υπολογισμούς που χρειάζονται για να βρεθεί σε πιο centroid αντιστοιχεί κάθε instance. Αυτοί οι υπολογισμοί αφορούν την εύρεση της απόστασης κάθε instance από όλα τα centroid και την σύγκριση όλων αυτών των τιμών. Αυτή η χρονική επιβάρυνση όπως βλέπουμε και από το Σχήμα 5.8 εκφράζεται γραμμικά

καθώς αυξάνουμε το πλήθος των cluster εκτός κάποιων μικρών ανωμαλιών σε κάποιες λίγες περιπτώσεις. Βλέπουμε πάλι εδώ αυτό που αναφέραμε και για τα dimension στη προηγούμενη ενότητα, ότι δηλαδή υπάρχει στενή αλληλεξάρτηση αυτών των δύο παραμέτρων και στον τρόπο που αυτές επηρεάζουν την χρονικη απόδοση καθώς αυξάνεται ο συντελεστής αναλογίας της εξάρτησης για αυξανόμενες τιμές των dimensions. Ισχύουν βέβαια και εδώ εντελώς αντίστοιχα οι παρατηρήσεις που κάναμε για τις χρονικές επιβαρύνσεις που προκύπτουν από τη διακίνηση δεδομένων μέσω δικτύου αλλά και αυτών που σχετίζονται με την χρήση του δίσκου.

Ανακεφαλαιώνοντας τα συμπεράσματά μας για την χρονική εκτέλεση μπορούμε συνολικά να πούμε ότι η αρχιτεκτονική εκτέλεσης του Spark καθώς και το caching συμβάλλουν καθοριστικά για την επίτευξη πολύ καλών χρονικών επιδόσεων. Η αύξηση του πλήθους των task η ικανοποίηση του data locality και εν γένει η διατήρηση της μέγιστης δυνατής παραλληλοποίησης κατά τη διάρκεια εκτέλεσης είναι μείζονα ζητήματα για τα κατανεμημένα συστήματα επεξεργασίας και το spark τα διαχειρίζεται με τον καλύτερο δυνατό τρόπο όπως φάνηκε από τη μελέτη μας για την εξάρτηση από το πλήθος των instances. Σημαντική επιβάρυνση παρατηρείται στην υπολογιστική καρδιά του αλγορίθμου όπου εκεί παίζει σημαντικό ρόλο το πλήθος των dimensions αλλά και των clusters. Η παράλληλη όμως συνεισφορά της παραμέτρου των dimensions τόσο στο πλήθος των task αλλά και στους υπολογισμούς την καθιστούν την πιο σημαντική όπως προέκυψε από τη μελέτη μας.

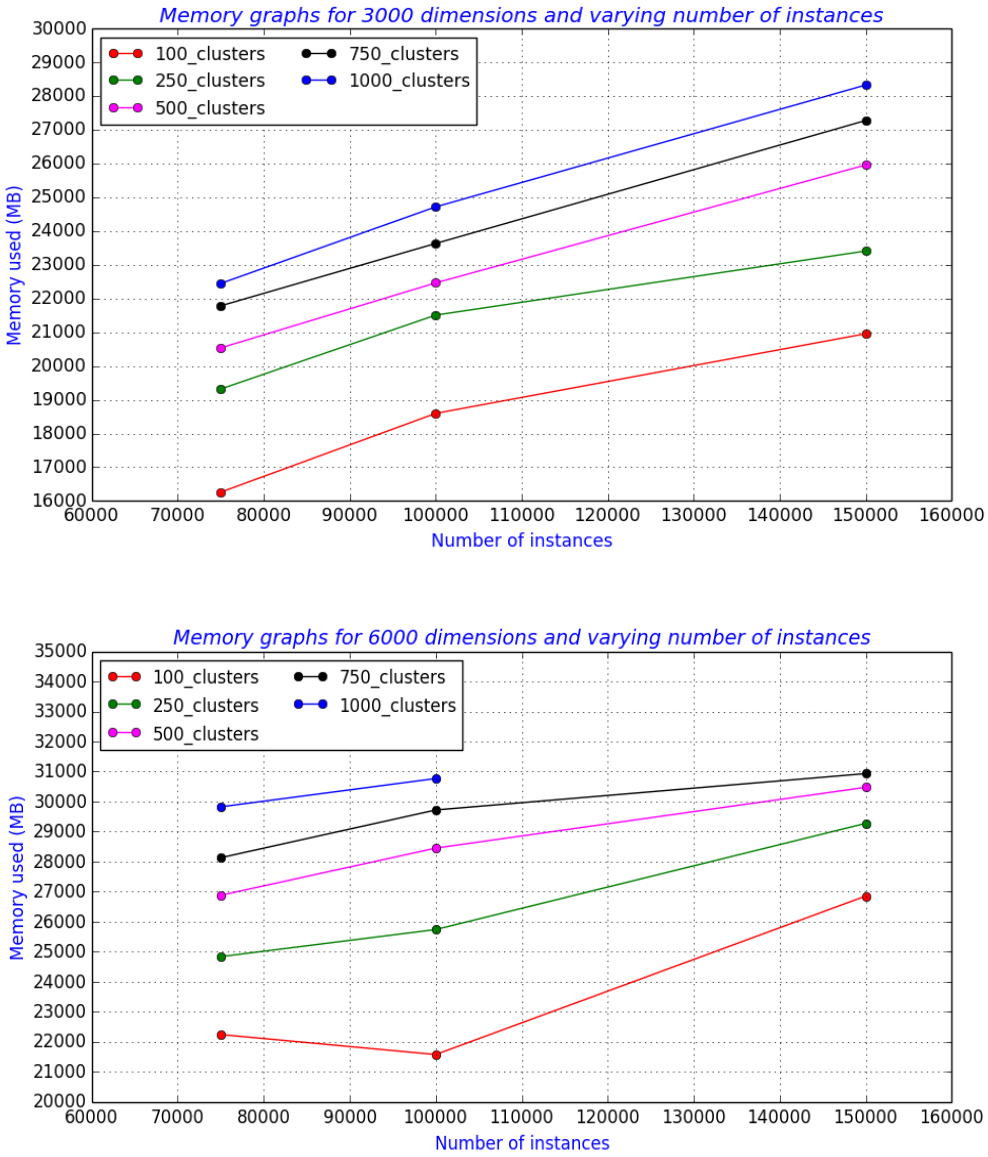
## 5.3.6 Χρήση μνήμης

Για τη μέτρηση της χρησιμοποιούμενης μνήμης υπολογίσαμε τη συνολική χρησιμοποιούμενη μνήμη κάθε εκτέλεσης αθροίζοντας τις επιμέρους τιμές που πήραμε για κάθε κόμβο του cluster. Οι μετρήσεις αυτές περιλαμβάνουν εκτός από το μέγεθος της μνήμης που δεσμεύουν οι executors του spark και το μέγεθος της μνήμης που χρησιμοποιούν οι διεργασίες datanode του hdfs και εκτελούνται στα ίδια μηχανήματα καθώς και οι διεργασίες worker, και αφορούν το μέσο όρο των τιμών που μετρήσαμε για τη μνήμη καθόλη τη διάρκεια εκτέλεσης. Η συνολική μνήμη του cluster που μελετούμε αποτελείται από αυτήν που διαθέτουν οι workers και υπολογίζεται 32GB και αυτή που διαθέτει ο master και είναι 8GB. Αναφέρουμε όμως εδώ ότι ο master δεν εκτελεί executor διεργασίες και άρα δεν λαμβάνει μέρος στην υπολογιστικό μέρος της εκτέλεσης, οπότε η χρήση της μνήμης του είναι πιο περιορισμένη όσο αφορά τις ανάγκες της εκτέλεσης.

Όπως είναι φυσικό η χρησιμοποίηση της διαθέσιμης μνήμης του cluster γίνεται σε συνάρτησει με το μέγεθος του dataset επί του οποίου εκτελείται ο αλγόριθμος. Αναμένουμε λοιπόν διαφοροποίηση των μετρούμενων τιμών κυρίως σε σχέση με διαφορετικές τιμές των dimensions και του πλήθους των instances και σχετικά σταθεροποιημένες τιμές για διαφορετικές τιμές cluster. Τα παραπάνω τα αναλύουμε διεξοδικότερα στις επόμενες τρεις υποενότητες που ακολουθούν.

### 5.3.6.1 Επίδραση πλήθους instances

Η εξάρτηση της μετρούμενης μνήμης που χρησιμοποιεί το cluster σε σχέση με το πλήθος των instances διαφοροποιείται για τις περιπτώσεις όπου έχουμε μικρότερα datasets ή μεγαλύτερα. Αυτό είναι αναμενόμενο δεδομένου ότι στη διάρκεια εκτέλεσης το spark διατηρεί, στο ποσοστό που του επιτρέπεται, το σύνολο του dataset στη μνήμη. Καθώς λοιπόν οι εκτελέσεις μας χρησιμοποιούν dataset που αυξάνονται σε μέγεθος τόσο περισσότερο πλησιάζουμε στο άνω όριο πληρότητας της διαθέσιμης μνήμης. Οι παραπάνω παρατηρήσεις συνοψίζονται στο Σχήμα 5.9 για δύο τέτοια dataset.



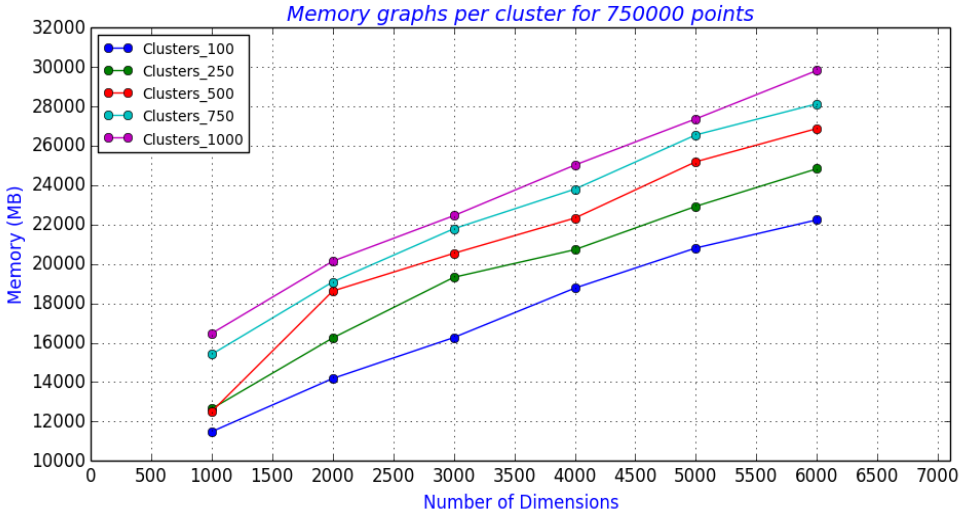

*Σχήμα 5.9: Χρήση μνήμης για διαφορετικό πλήθος instances*

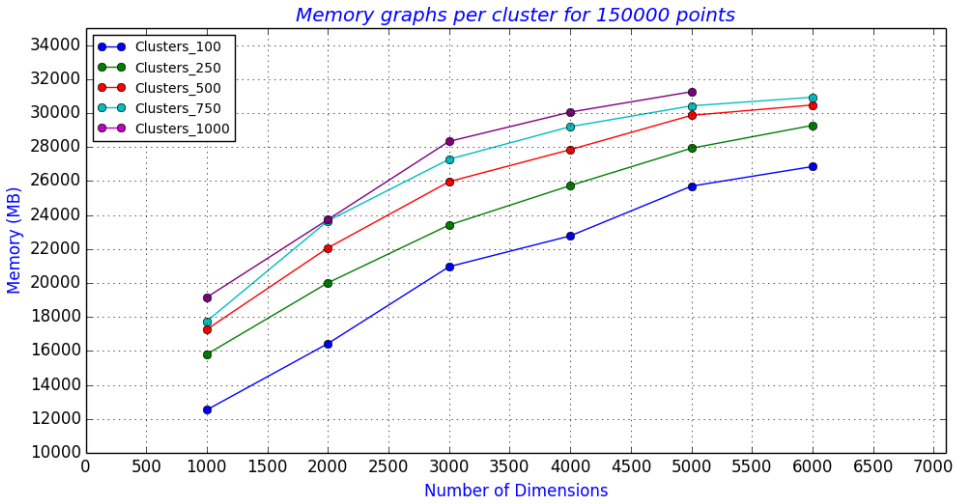
Στη περίπτωση των καμπυλών με 3000 dimensions το μέγεθος του dataset για κάθε τιμή των cluster είναι 4.5,6 και 9GB αντίστοιχα καθώς αυξάνουμε την τιμή του πλήθους των instances. Η ανταπόκριση της μετρούμενης τιμής σε αυτή την αύξηση είναι μία εντελώς γραμμική αντίστοιχα αύξησή της με πολύ μικρές αποκλίσεις. Αυτές οι μετρήσεις ανταποκρίνονται ακριβώς στην όλο και μεγαλύτερη κάλυψη της cache του cluster από το dataset. Αντίθετα στη περίπτωση των καμπυλών με τα 6000 dimensions έχουμε αντίστοιχα

μεγέθη dataset 9,12 και 18GB για τις τιμές πληθους instances 75000, 100000 και 150000 αντίστοιχα. Σε αυτές βλέπουμε μία πολύ ομαλότερη αύξηση της παρατηρούμενης τιμή κυρίως μετά από τα 100000 instances. Αυτό γιατί από εκείνη την τιμή και μετά έχουμε πλήρωση της cache του cluster και καθώς αυξάνεται το μέγεθος του dataset η χρήση μνήμης αγγίζει τα όριά της που είναι τα 32GB. Και πάλι όπως και σε προηγούμενες γραφικές αναφέρουμε εδώ την απουσία της τιμής για 150000 instances και 6000 dimensions για τα 1000 cluster η οποία δεν εκτελέστηκε επιτυχώς.

### 5.3.6.2 Επίδραση πλήθους dimensions

Το πλήθος των dimensions είναι ο έτερος σημαντικό παράγοντας που επηρεάζει τη χρήση μνήμης στο cluster κατά την εκτέλεση. Η συνεισφορά του στο μέγεθος ενός dataset έχει άμεσο αντίκτυπο στο τι ποσοστό καταλαμβάνει αυτό στη cache του cluster και άρα τι ποσοστό της συνολικής μνήμης βρίσκεται σε χρήση. Στο Σχήμα 5.10 βλέπουμε τη χρήση της μνήμης στο cluster για δυο διαφορετικές περιπτώσεις πλήθους instances και για όλες τις τιμές cluster.





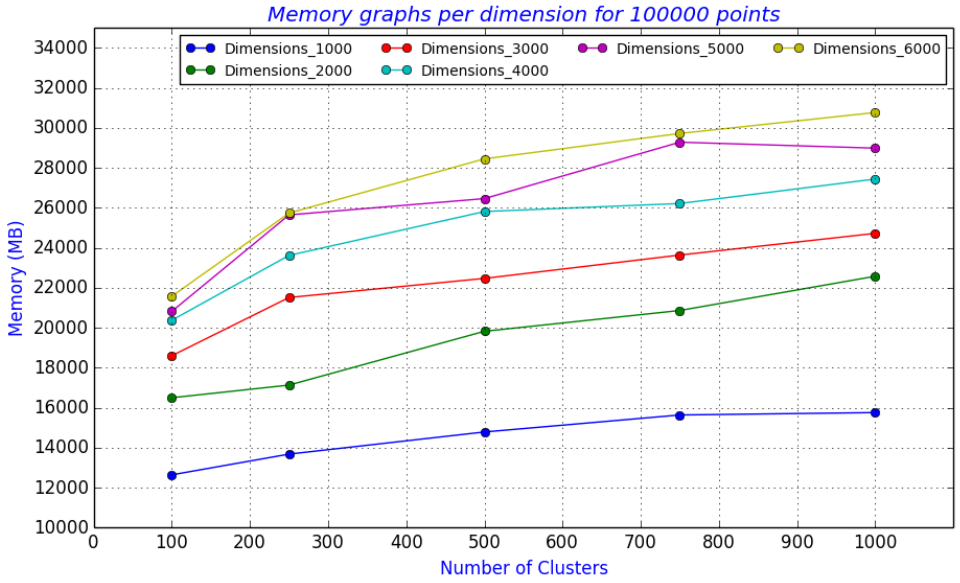

*Σχήμα 5.10: Χρήση μνήμης για διαφορετικά dimensions*

Στις καμπύλες της πρώτης γραφικής τα μεγέθη των dataset είναι όλα ,συγκρινόμενα με την cache του cluster, μικρότερου μεγέθους. Σε αυτή τη περίπτωση βλέπουμε μία πλήρως γραμμική εξάρτηση της μνήμης από την αύξηση του πλήθους των dimensions η οποία εκφράζεται και με ένα σχετικά σταθερό βήμα 2GB καθώς αυξάνουμε τα dimensions κατά 1000 σε κάθε εκτέλεση. Στις περιπτώσεις της δεύτερης γραφικής έχουμε αρκετά μεγαλύτερα dataset για κάθε εκτέλεση καθώς αυξάνουμε τα dimensions και για αυτό παρατηρούμε τον κορεσμό της συνολικής μνήμης του cluster καθώς πλησιάζουμε πιο κοντά στη τιμή των 6000 για τα dimensions.

### 5.3.6.3 Επίδραση πλήθους cluster

Το πλήθος των cluster δεν αναμένουμε να έχει μεγάλη επίπτωση στο μέγεθος της χρησιμοποιούμενης μνήμης σε κάθε εκτέλεση γιατί δεν αποτελεί παράμετρο που επηρεάζει το μέγεθος του dataset. Παρόλα αυτά υπενθυμίζουμε από την εισαγωγική ενότητα 2.2.3.4 όπου είχαμε αναλύσει την χρήση της cache από το spark ότι υπάρχει ένα ποσοστό της cache που προορίζεται ειδικά για χρήση των δεδομένων shuffle write που προκύπτουν από το τέλος κάθε mapPartiotions stage. Εκεί είχαμε πει ότι το ποσό της μνήμης που διατίθεται για την παραπάνω χρήση είναι το 20% της safe region του heap που έχει κάθε executor. Αυτό το ποσοστό δεδομένου ότι αφορά αποκλειστικά τα δεδομένα shuffle write παραμένει ανεκμετάλλευτο στη περίπτωση που αυτά είναι μικρότερου μεγέθους.

Όπως έχουμε ήδη αναφέρει η παράμετρος cluster, μαζί βέβαια και με τα dimensions, είναι αυτή που επηρεάζει το μέγεθος των δεδομένων shuffle write άρα και το ποσοστό της παραπάνω διαθέσιμης μνήμης cache που χρησιμοποιείται κατά την εκτέλεση για την αποθήκευσή τους. Με βάση τα παραπάνω η ακριβέστερη διατύπωση είναι ότι η μεταβολή της χρησιμοποιούμενης μνήμης παραμένει σχεδόν σταθερή τουλάχιστον μετά το πέρας μιας τιμής cluster που κατασκευάζει shuffle write δεδομένα που πληρούν το αντίστοιχο απόθεμα cache μνήμης. Τα παραπάνω εκφράζονται γραφικά για την περίπτωση 100000 instances και για όλα τα πιθανά dimensions στο Σχήμα 5.11.



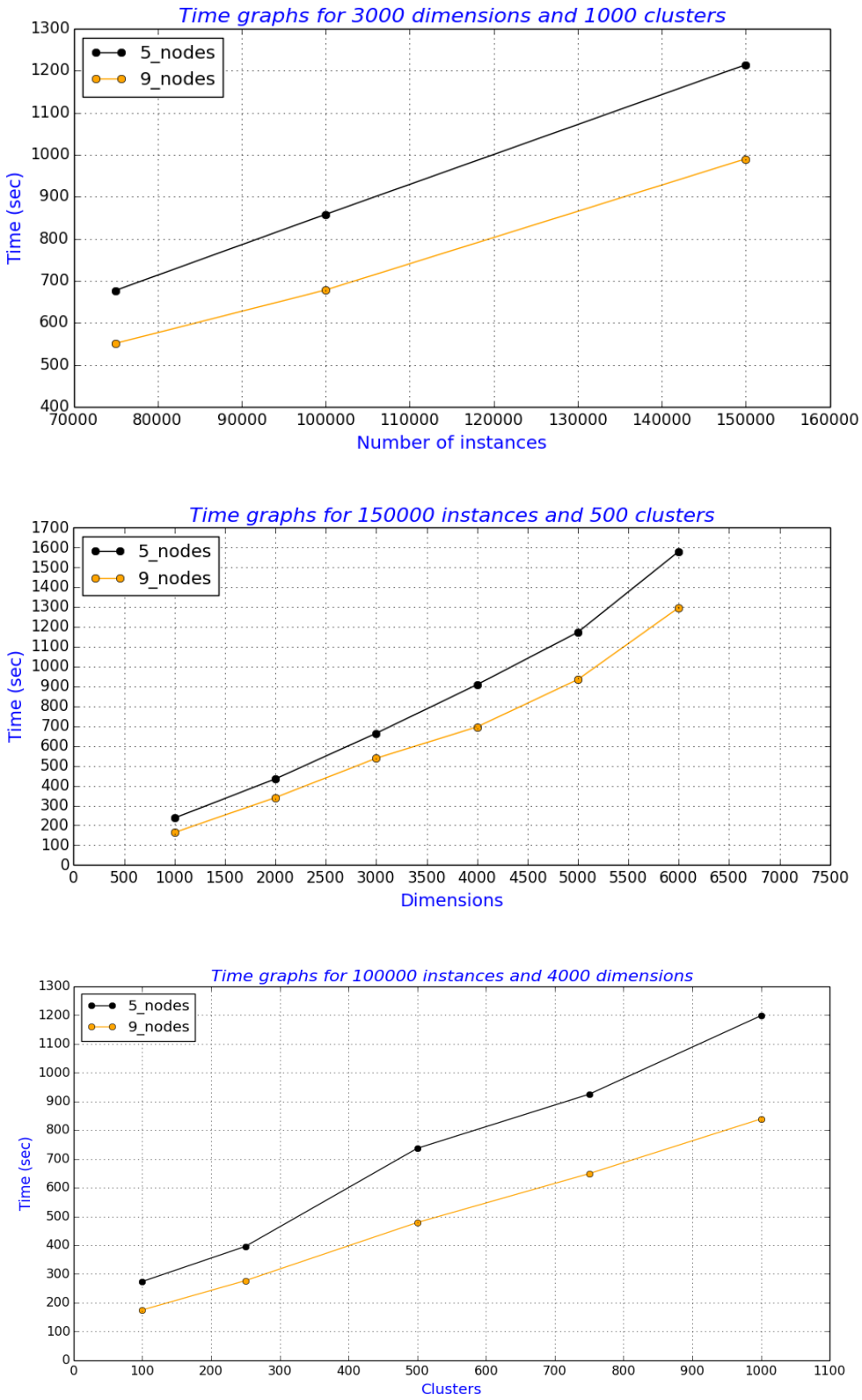

*Σχήμα 5.11: Χρήση μνήμης για διαφορετικά cluster*

### 5.3.7 Σύγκριση με διαφορετικά nodes ανά cluster

Για τη σύγκριση εκτελέσεων μεταξύ διαφορετικών μεγεθών cluster, χρησιμοποιήσαμε το cluster με τους 5 nodes (1 master, 4 workers) που είδαμε και εισαγωγικά σε αυτό το κεφάλαιο. Σε αυτό το cluster διατηρήσαμε την ίδια συνολική μνήμη με την περίπτωση των 9 nodes αλλά και τα ίδια cores ανά node. Οι παραπάνω μετατροπές δεν αλλάζουν την ικανότητα του cluster να χρησιμοποιεί μεγαλύτερη cache ενώ επίσης ο κάθε node έχει την ίδια υπολογιστική δυναμική να εκτελεί δύο tasks σε παραλληλία. Με αυτό τον τρόπο θα συγκρίνουμε μόνο την κλιμακωσιμότητα του αλγορίθμου σε συνάρτησει με το πλήθος των nodes του cluster.

Εφ'όσον επιλέξαμε η συνολική μνήμη του μικρότερου αυτού cluster να είναι η ίδια με το μεγαλύτερο, για τη χρήση μνήμης παρατηρήσαμε ότι η συμπεριφορά και η μετρούμενες τιμές έχουν μικρή διαφοροποίηση από αυτά που είδαμε στην ενότητα 5.3.6. Για τον ίδιο λόγο παρατηρήσαμε ίδιες τιμές και για τα μεγέθη disk input και disk output του cluster με επίσης μικρές αποκλίσεις. Παρόλα αυτά οι παραπάνω μετρήσεις αποδείχθηκαν χρήσιμες στο να επιβεβαιώσουν την συμπεριφορά και τα συμπεράσματα που εξάγαμε στις αντίστοιχες ενότητες για το μεγαλύτερο cluster υποδηλώνοντας τον ενιαίο χαρακτήρα εκτέλεσης του αλγορίθμου ασχέτως μεγέθους cluster.

Αντιθέτως για τη χρονική επίδοση των εκτελέσεών μας παρατηρήσαμε σημαντική διαφοροποίηση για τα δύο αυτά διαφορετικά cluster η οποία αποτυπώνεται για διαφορετικές τιμές instances, dimensions και cluster στο Σχήμα 5.12.




*Σχήμα 5.12: Σύγκριση χρονικής επίδοσης για διαφορετικά nodes ανά cluster*

Αυτό που φαίνεται από το σχήμα 5.12 είναι η χρονική υπεροχή του cluster με τα 9 nodes σε κάθε εκτέλεση. Παρότι εδώ δείχνουμε μεμονωμένες περιπτώσεις αυτή η παρατήρηση ισχύει για κάθε εκτέλεση του αλγορίθμου που επιχειρήσαμε. Για όλες αυτές τις

περιπτώσεις υπολογίσαμε την χρονική βελτίωση καθώς αυξήθηκε το πλήθος των nodes ανά cluster ποσοστιαία και βρήκαμε μία βελτίωση 22.75% κατά μέσο όρο με μερικές περιπτώσεις εκτελέσεων να αγγίζουν και χρονική βελτίωση της τάξης του 35%. Η θεωρητική ιδανική ποσοστιαία χρονική βελτίωση θα έπρεπε να αγγίζει το 50% (υποδιπλασιασμός του χρόνου) δεδομένου ότι διπλασιάζουμε το μέγεθος του cluster και ουσιαστικά διπλασιάζουμε την υπολογιστική του ικανότητα και την δυνατότητα παραλληλοποίησης. Αυτή η βελτίωση στη πράξη δύσκολα επιτυγχάνεται λόγω μιας πληθώρας παραγόντων που επηρεάζουν ποικιλοτρόπως τις εκάστοτε εκτελέσεις, παρόλα αυτά το ποσοστό του 22.75% είναι αρκετά ικανοποιητικό και εκφράζει μια καλή κλιμακωσιμότητα του αλγορίθμου σε σχέση με τη χρονική επίδοση.

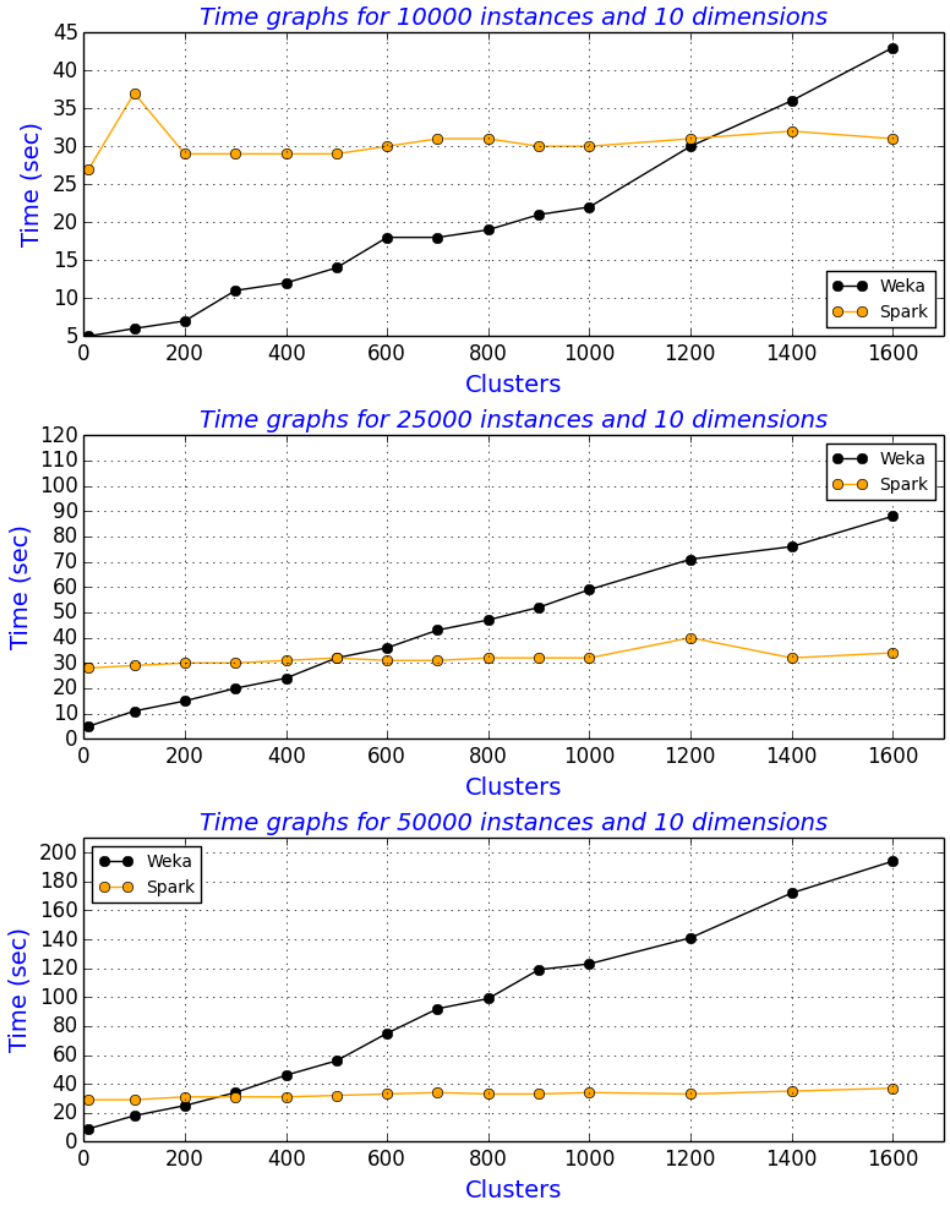
Αυτό που πυροδοτεί αυτή τη βελτίωση μεταξύ των 2 περιπτώσεων είναι σε μεγάλο βαθμό το πλήθος των task που μπορούν να δρομολογηθούν παράλληλα στους nodes του cluster. Η περίπτωση των 8 workers παρέχει τη δυνατότητα παράλληλης εκτέλεσης 16 task στο cluster ενώ η περίπτωση των 4 workers δίνει την δυνατότητα παράλληλης εκτέλεσης 8 task. Αυτή η διαφορά φαίνεται καλύτερα στη τελευταία καμπύλη που γίνεται σύγκριση μεταξύ dataset που έχουν το ίδιο μέγεθος και άρα συνολικά χρειάζεται να δρομολογηθεί ίδιο πλήθος task. Η διαφορά που παρατηρούμε από εκτέλεση σε εκτέλεση με διαφορετικά cluster αντικατοπτρίζει ακριβώς την δυνατότητα μεγαλύτερης παραλληλοποίησης που προσφέρει το μεγαλύτερο cluster σε σχέση με το μικρότερο. Αυτή η διαφορά μάλιστα εντείνεται καθώς μεγαλώνει η τιμή των cluster και έτσι αυξάνεται η χρονική διάρκεια που χρειάζεται κάθε task να εκτελεστεί.

### 5.3.8 Σύγκριση υλοποιήσεων Spark και Weka

Για τους σκοπούς της σύγκρισης της κεντρικής υλοποίησης με τη κατανεμημένη υλοποίηση πραγματοποιήσαμε όλες τις εκτελέσεις με τις οποίες στηριχθήκαμε για την ανάλυσή μας στο κεφάλαιο 4. Η μελέτη της σύγκρισης αποσκοπεί στο να δούμε πότε και κάτω υπο ποιες προϋποθέσεις υπερτερεί η μία υλοποίηση συγκριτικά με την έταιρη. Είναι προφανές ότι για μεγάλα dataset αυτή η σύγκριση στερείται νοήματος αφού η κατανεμημένη προσφέρει πολύ μεγαλύτερη υπολογιστική ισχύ για αυτό επικεντρωνόμαστε σε εκείνες τις περιπτώσεις που τα dataset διατηρούνται σε μικρά μεγέθη.

Όσον αφορά το ποιές μετρήσεις από αυτές που συγκεντρώσαμε μπορούμε να συγκρίνουμε ανάμεσα στις δύο υλοποιήσεις αυτή είναι μονάχα η χρονική επίδοση του αλγορίθμου για διαφορετικές παραμέτρους εκτέλεσης. Για τη μνήμη είναι σαφές ότι δεν μπορούμε να συγκρίνουμε τη συνολική μνήμη ενός cluster με αυτή που χρησιμοποιείται από ένα κεντρικό κόμβο. Για το disk output μπορούμε απλά να παρατηρήσουμε στη κεντρική υλοποίηση ότι δεν έχουμε καμία εγγραφή στο δίσκο κατά τη διάρκεια εκτέλεσης του αλγορίθμου ενώ αντίθετα για την κατανεμημένη υλοποίηση οι εγγραφές στο δίσκο πάντα ενυπάρχουν έστω και σε πολύ μικρές τιμές γιατί αυτές σχετίζονται με το shuffle write. Όσον αφορά τη μέτρηση του disk input αυτή στη κεντρική υλοποίηση αλλά και στη κατανεμημένη είναι ισόποση γιατί αφορά καθαρά το μέγεθος του dataset που διαβάζουν οι δύο υλοποιήσεις κατά την εκκίνησή τους.

Για τη σύγκριση της χρονικής επίδοσης παραθέτουμε στο Σχήμα 5.13 τρεις γραφικές για διαφορετικά πλήθη instances και μεταβαλλόμενο αριθμό cluster. Αυτές επιλέχθηκαν με κριτήριο να δειχθεί για ποιες τιμές των παραμέτρων είναι εφικτή η χρονική σύγκριση και για ποιες ξεκινάει να γίνεται εμφανής η διαφορά.




*Σχήμα 5.13: Σύγκριση χρονικής επίδοσης κεντρικής και κατανεμημένης υλοποίησης*

Υπενθυμίζουμε ότι οι εκτελέσεις στο Spark που φαίνονται στο Σχήμα 5.13 αφορούν το cluster για τους 9 nodes. Επειδή τα datasets είναι πολύ μικρά για την κατανεμημένη υλοποίηση αυτό που συμβαίνει είναι ότι στο Spark το dataset χωρίζεται σε τόσα partitions όσα είναι και τα cores του cluster και εκτελεί πλήρως παράλληλα κάθε επανάληψη του αλγορίθμου. Επειδή δε αυτά τα partitions είναι επίσης πολύ μικρά το κάθε task που εκτελείται διαρκεί πάρα πολύ λίγο, άρα ουσιαστικά η κυρίαρχη χρονική καθυστέρηση προέρχεται από τη διαδικασία διαβάσματος του dataset από το hdfs, τη διαδικασία

εκπόνησης φυσικού πλάνου εκτέλεσης από τη μεριά του driver και τη δρομολόγηση των task στους nodes του cluster. Η τελευταία παραμένει περίπου σταθερή για όλες τις εκτελέσεις περίπου στα 30 sec και άρα η κεντρική υλοποίηση είναι μόνο τότε καλύτερη όταν καταφέρνει να ολοκληρώσει την εκτέλεσή της κάτω από αυτό το όριο.

Το παραπάνω φαινόμενο παρατηρείται μόνο για πολύ μικρά dataset της τάξης των μερικών MB. Αυτό σημαίνει ότι πρέπει να ενυπάρχει ταυτόχρονος περιορισμός και των dimensions και του πλήθους των instances αλλά και του πλήθους των cluster όπου όπως βλέπουμε από την αύξησή του υποφέρει η κεντρική υλοποίηση. Μπορούμε ίσως να πούμε ότι η κεντρική υλοποίηση είναι προτιμητέα αν χαλαρώσουμε τη απαίτησή μας να είναι αυστηρώς καλύτερη χρονικά και συνυπολογίσουμε το γεγονός ότι είναι πολύ πιο απλή στην εκτέλεσή της. Με αυτό το σκεπτικό ίσως οι διαφοροποιήσεις στο χρόνο που βλέπουμε επί παραδείγματι για τις τιμές των 50000 instances για τα μεγάλα cluster να είναι ανεκτές.

Αυτό που συνολικά μπορούμε να πούμε για τις δύο εκτελέσεις από την εμπειρία που αποκτήσαμε με την ενασχόληση μαζί τους είναι δύο πράγματα. Το πρώτο έχει να κάνει με την κεντρική υλοποίηση του weka που υποφέρει από το γεγονός ότι δεν είναι παραλληλοποιήσιμη. Το γεγονός ότι δεν κάνει χρήση όλων των διαθέσιμων cores ενός μηχανήματος την καθιστά πολύ γρήγορα μη ανταγωνιστική και την περιορίζει σημαντικά. Ίσως μια παράλληλη εκδοχή να ήταν καλύτερη σε ένα μεγαλύτερο έυρος περιπτώσεων από αυτές που είδαμε τώρα. Το δεύτερο έχει να κάνει με την απόδοση του Spark σχετικά με τον τρόπο που πραγματοποιεί την κατάτμηση μιας εφαρμογής σε task, την ταχύτητα του scheduler που διαθέτει, και γενικότερα την επιτυχία του να περιορίζει τα χρονικά κόστη που δεν έχουν άμεση σχέση με την εκτέλεση του αλγορίθμου όπως είναι για παράδειγμα και η χρήση του δικτύου. Τα παραπάνω αν και έχουν τα έχουμε αναφέρει και σε προηγούμενες ενότητες γίνονται ακόμα πιο καθαρά αντιληπτά στις περιπτώσεις του Σχήματος 5.13.

## 5.4 Μοντελοποίηση

Όπως είδαμε από τη μέχρι τώρα ανάλυσή μας και τις μετρικές που λάβαμε για τις εκτελέσεις στο Spark ιδιαίτερο ενδιαφέρον παρατηρούν αυτές που αφορούν το disk output, την χρησιμοποιούμενη μνήμη του cluster και το χρόνο εκτέλεσης. Για τη περίπτωση του disk input η συμπεριφορά του συστήματος είναι προβλέψιμη για datasets που είναι μικρότερα της συνολικής cache ενώ για τις υπόλοιπες περιπτώσεις δεν έχουμε ικανό αριθμό δειγμάτων-εκτελέσεων που να μας επιτρέψουν να δημιουργήσουμε αξιόπιστα μοντέλα για αυτή την μέτρηση. Για τους λόγους αυτούς δεν θα ασχοληθούμε με τη μοντελοποίηση αυτής της μετρικής στη παρούσα εργασία, θα πρέπει να αναφέρουμε όμως ότι δεν στερείται ενδιαφέροντος και σημασίας μελέτης σε σχέση με τις υπόλοιπες

Στα επόμενα αφιερώνουμε μία ενότητα ανά μετρική που μας ενδιαφέρει και προσπαθούμε να κατασκευάσουμε μοντέλα που να περιγράφουν την μετρική με τη μεγαλύτερη δυνατή ακρίβεια. Για το σκοπό αυτό όπως και στη κεντρική υλοποίηση πειραματιστήκαμε με μία σειρά μοντέλων που προσφέρει το weka τα οποία εκπαιδεύσαμε και συγκρίναμε τα αποτελέσματά τους. Στις ενότητες που ακολουθούν παρουσιάζουμε μόνο αυτά που είχαν την πιο ικανοποιητική απόδοση, εκθέτουμε τα σφάλματα που αυτά παρουσιάζουν, δίνουμε γραφικά την προσέγγισή τους και πραγματοποιούμε συγκριτική

ανάλυση όπου αυτό είναι δυνατό. Όλα τα δεδομένα που χρησιμοποιήσαμε προκύπτουν από τις συνολικά 89 εκτελέσεις που πραγματοποιήσαμε για την κατανεμημένη μελέτη μας στο cluster με τους 9 nodes. Εξαιρούμε παντού την ενενηκοστή μέτρηση που αφορά εκτέλεση που αδυνατούσε το spark να εκτελέσει όπως έχουμε ήδη δει.

### 5.4.1 Μοντελοποίηση disk output

Ένα δείγμα των δεδομένων που χρησιμοποιήσαμε για τη μοντελοποίηση της μέτρησης disk output φαίνεται στο Πίνακα 5.3 παρακάτω.

| Instances | Dimensions | Clusters | Disk output (MB) |
|---|---|---|---|
| 75000 | 1000 | 250 | 200 |
| 100000 | 3000 | 500 | 5799 |
| 150000 | 5000 | 750 | 32596 |

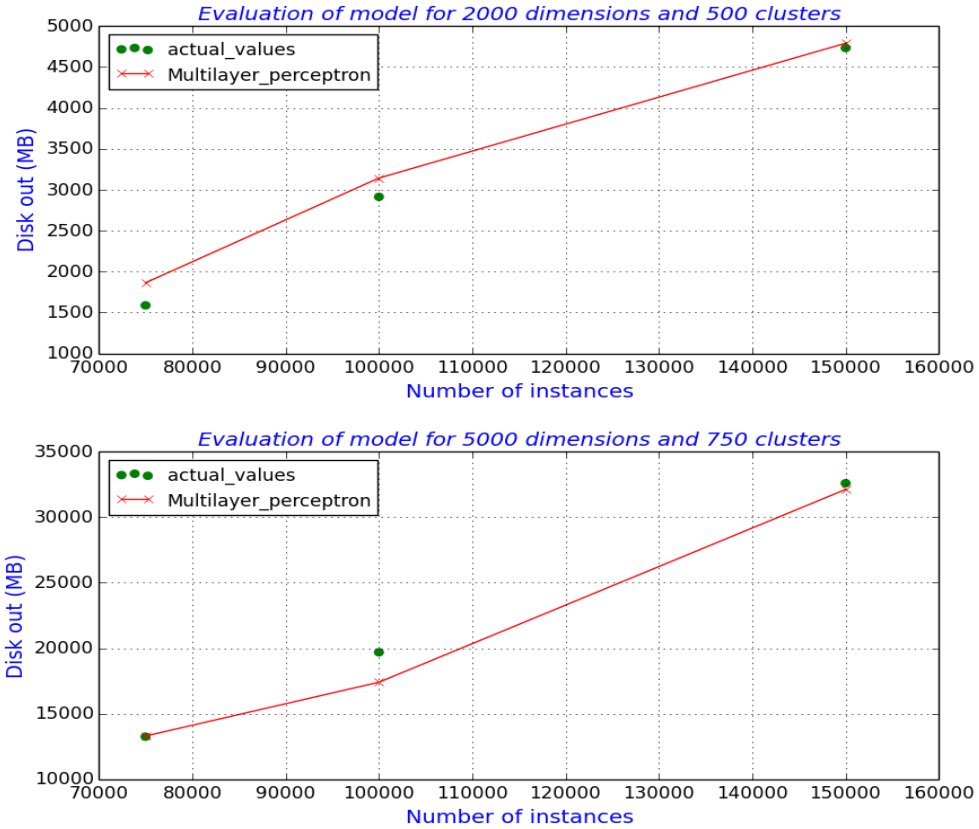
*Πίνακας 5.3: Δείγμα δεδομένων μοντελοποίησης disk output*

Ο πειραματισμός μας με ένα πλήθος μοντέλων του weka, μεταξύ των οποίων και αυτά που ήδη έχουμε δει από την αντίστοιχη μελέτη μας στη κεντρική υλοποίηση δεν προσέφεραν ικανοποιητική προσέγγιση της μετρικής μας ώστε να συμπεριληφθούν στη παρουσίασή μας. Εξαίρεση αποτέλεσε η χρησιμοποίηση του Multilayer Perceptron μοντέλου το οποίο με χρήση συγκεκριμένων παραμέτρων όπως ο ρυθμός μάθησης (learning rate), το momentum και το πλήθος κόμβων του νευρωνικού δικτύου παρέχει πολύ καλή ακρίβεια με σφάλματα που φαίνονται στο Πινακα 5.4 για training set 10% επί του συνόλου των δεδομένων που διαθέσαμε.

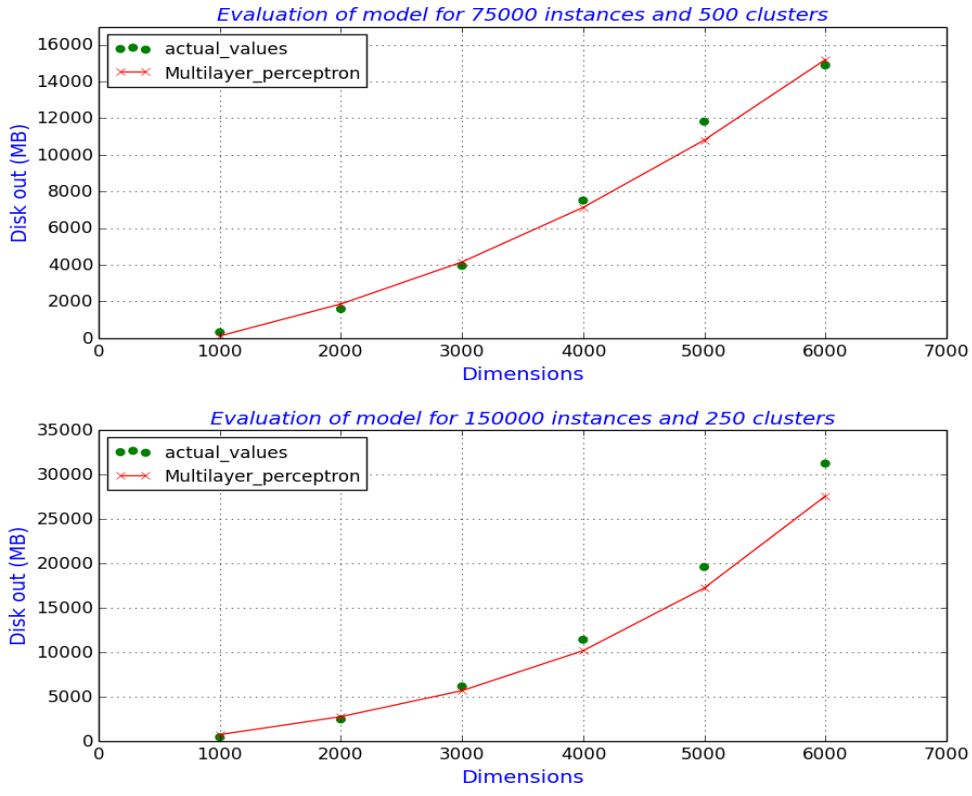
| Model | Mean absolute error (MB) | Mean relative error |
|---|---|---|
| Multilayer Perceptron | 575 | 6.32% |

*Πίνακας 5.4: Σφάλματα μοντελοποίησης disk output*

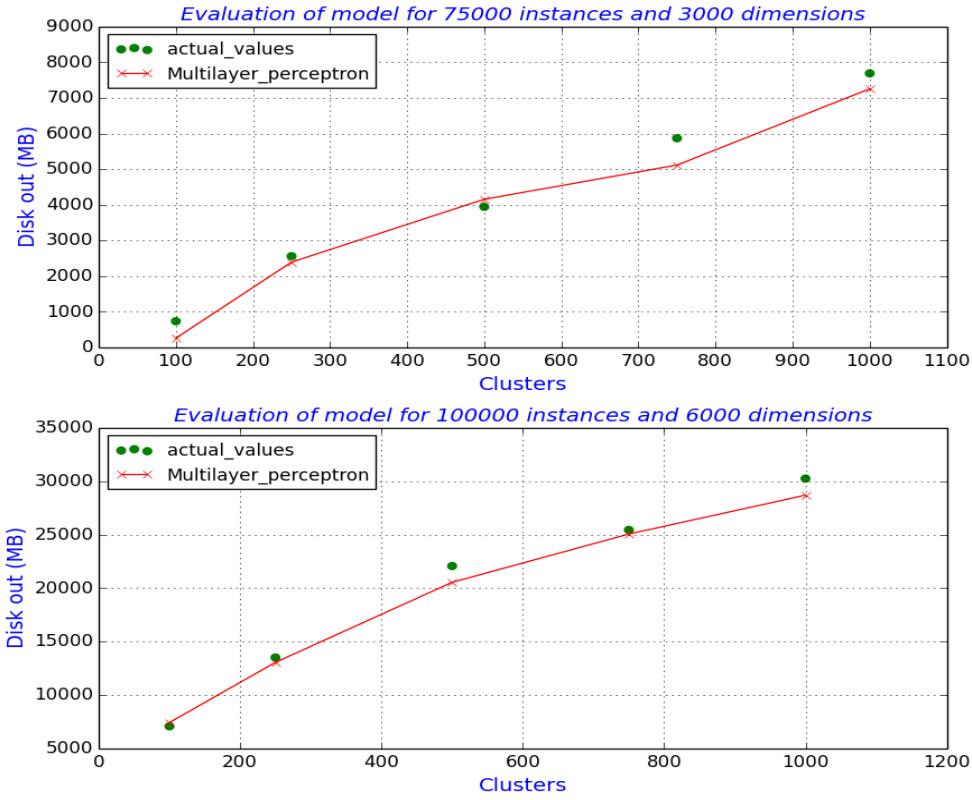
Η αδυναμία προσέγγισης της μετρικής με διαφορετικά μοντέλα μπορεί να οφείλεται σε μια πληθώρα παραγόντων. Ένας από αυτούς είναι η ιδιαίτερη συμπεριφορά που παρουσιάζει η μετρική του disk output που όπως είδαμε αυτή επηρεάζεται ποικιλοτρόπως από τις παραμέτρους εκτέλεσης με μεγάλο εύρος τιμών. Το παραπάνω γεγονός σε συνδυασμό με το περιορισμένο πλήθος δεδομένων που μπορούμε να διαθέσουμε λόγω της φύσης του προβλήματος απαιτεί πολύ ισχυρά και εκφραστικά μοντέλα όπως αυτά είναι τα νευρωνικά δίκτυα. Τα αποτελέσματα των σφαλμάτων του Πίνακα 5.4 είναι πολύ ενθαρρυντικά σε πρώτη φάση για την κατασκευή ενός μοντέλου ισχυρού, παραθέτουμε όμως παρακάτω και τη γραφική σκοπιά αυτής της προσέγγισης στα Σχήματα 5.14, 5.15 και 5.16 όπου σε κάθε ένα από αυτά μεταβάλλουμε μία μόνο παράμετρο εκτέλεσης και παραθέτουμε τις τιμές της προβλεπτικής μοντελοποίησης καθώς και τις πραγματικές μετρήσεις για επιλεγμένες περιπτώσεις κάθε φορά.




*Σχήμα 5.14: Μοντελοποίηση disk output για μεταβαλλόμενο πλήθος instances*




*Σχήμα 5.15: Μοντελοποίηση disk output για μεταβαλλόμενο πλήθος dimensions*




*Σχήμα 5.16: Μοντελοποίηση disk output για μεταβαλλόμενο πλήθος cluster*

Η γραφική επαλήθευση του μοντέλου που βλέπουμε στα παραπάνω σχήματα συνάδει και με τις πολύ καλές τιμές σφαλμάτων που είδαμε. Αυτό που έχει αξία παρατήρησης είναι η δυνατότητα του Multilayer Perceptron να παρεμβάλλεται με μεγάλη ακρίβεια σε χώρους που η εναλλαγή τιμών γίνεται με πολύ διαφορετικό τρόπο. Συγκεκριμένα βλέπουμε εδώ ότι οι πραγματικές τιμές εναλλάσονται ανάλογα με τη μεταβολή της μετρικής ανά γραφική, αρχικά με ένα γραμμικό τρόπο σε σχέση με το πλήθος των instances, μετά με παραβολικό τρόπο σε σχέση με τα dimension και με ένα σχεδόν γραμμικό τρόπο σε σχέση με τα cluster. Σε όλες τις παραπάνω περιπτώσεις το μοντέλο ανταποκρίνεται με μεγάλη ακρίβεια παρέχοντας πολύ ισχυρή προβλεπτική μοντελοποίηση.

Η απουσία εναλλακτικών επιλογών μοντέλων για τη συγκεκριμένη μετρική σε συνδυασμό με την πολύ καλή εναλλακτική του νευρωνικού δικτύου, μας οδήγησε να σκεφτούμε μήπως η κατασκευή του μοντέλου μας είναι αποτέλεσμα overfitting. Το overfitting είναι αποτέλεσμα υπερεκπαίδευσης του μοντέλου με συγκεκριμένα δεδομένα και απουσία γενίκευσης των αποτελεσμάτων του για εναλλακτικά δεδομένα. Στη περίπτωση αυτή το μοντέλο παρέχει πολύ καλή προσέγγιση για τα δεδομένα με τα οποία γίνεται η εκπαίδευσή του και αδυνατεί να κάνει καλή προσέγγιση σε νέα παρεχόμενα δεδομένα. Συνήθως αυτό προκαλείται είτε από training sets τα οποία δεν είναι ενδεικτικά και δεν περιγράφουν στην ολότητά της μια διαδικασία είτε επειδή επιλέγεται μεγάλος αριθμός εποχών εκπαίδευσης που ορίζει το πλήθος των φορών που τροφοδοτούμε το νευρωνικό δίκτυο ώστε να συγκλίνει. Το πρώτο από τα παραπάνω δύο ενδεχόμενα δεν είναι δυνατό να συμβαίνει στη περίπτωσή μας διότι και από τη θεωρητική γνώση εκτέλεσης του αλγορίθμου μπορούμε με σχετική βεβαιότητα να συμπεράνουμε ότι το training set περιέχει δεδομένα

που ανταποκρίνονται σε όλες τις πιθανές συνθήκες εκτέλεσης. Όσον αφορά το πλήθος των εποχών εκπαίδευσης του νευρωνικού πειραματιστήκαμε με αρκούντως μικρές τιμές και τα αποτελέσματα ήταν επίσης πολύ καλά σχετικά με την ακρίβεια του μοντέλου. Τα παραπάνω επιβεβαίωσαν σε μεγάλο βαθμό την ποιότητα της μοντελοποίησης που επιχειρήσαμε.

### 5.4.2 Μοντελοποίηση χρόνου εκτέλεσης

Για την μοντελοποίηση του χρόνου εκτέλεσης των εκτελέσεων που επιχειρήσαμε παραθέτουμε ένα δείγμα των δεδομένων μοντελοποίησης στον Πίνακα 5.5 παρακάτω.

| Instances | Dimensions | Clusters | Time (sec) |
|---|---|---|---|
| 75000 | 3000 | 100 | 122 |
| 100000 | 4000 | 500 | 479 |
| 150000 | 6000 | 750 | 1704 |

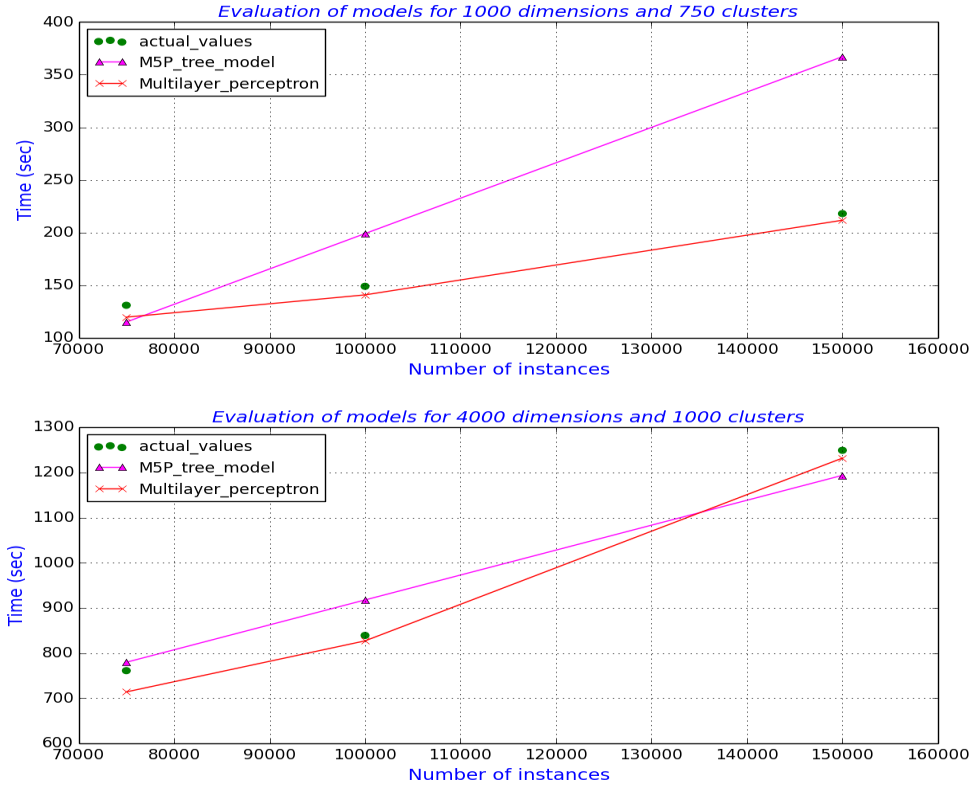
*Πίνακας 5.5: Δείγμα δεδομένων μοντελοποίησης χρόνου εκτέλεσης*

Εντελώς αντίστοιχα με τις προηγούμενες περιπτώσεις μοντελοποίησης χρησιμοποιήσαμε πολλά διαφορετικά μοντέλα και καταλήξαμε ότι την καλύτερη προσέγγιση προσφέρουν το Multilayer Perceptron και το M5P tree model. Για τα δύο αυτά μοντέλα σύμφωνα με την προσφιλή μας μέχρι τώρα τακτική χρησιμοποιήσαμε το ποσοστό 10% για test set και το 90% για training set. Τα σφάλματα που προέκυψαν από τη διαδικασία φαίνονται στον Πίνακα 5.6 παρακάτω.

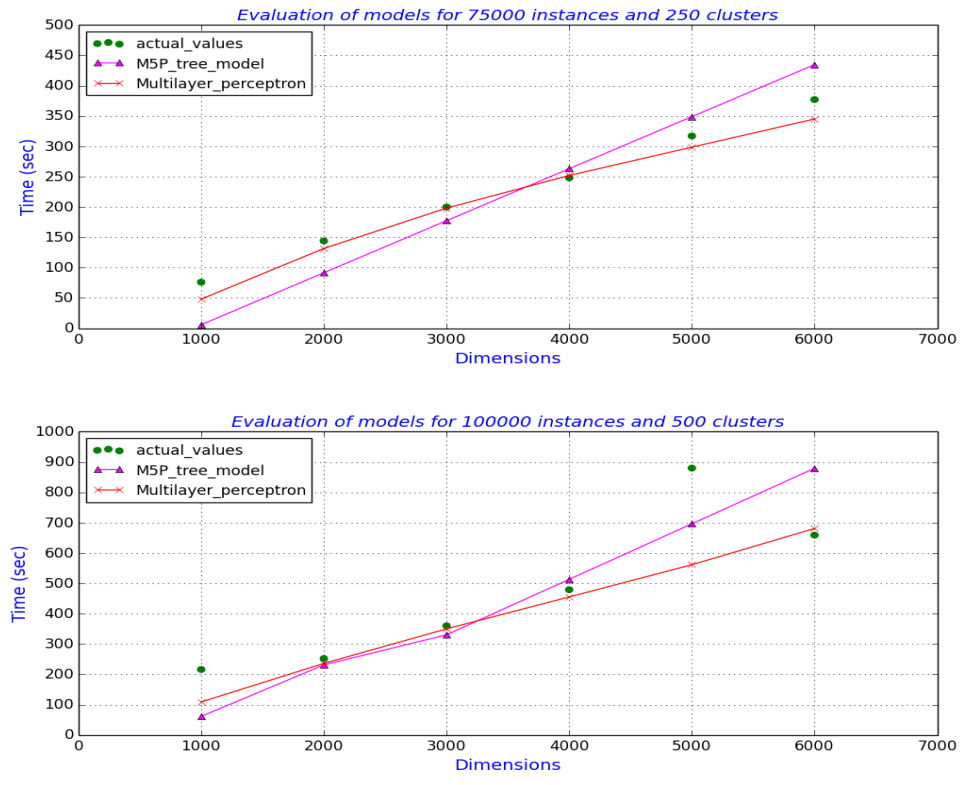
| Models | Mean Absolute Error (sec) | Mean Relative Error |
|---|---|---|
| M5P tree model | 131 | 40,24% |
| Multilayer Perceptron | 47 | 14,41% |

*Πίνακας 5.6: Σφάλματα μοντελοποίησης χρόνου εκτέλεσης*

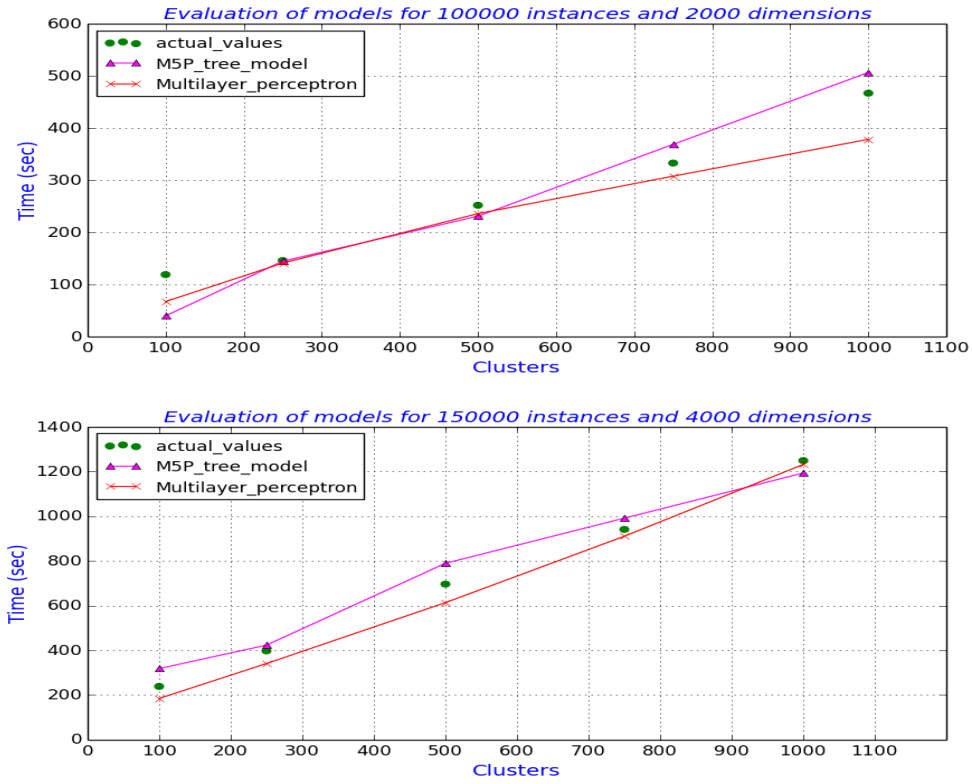
Οι υπολογισμοί για τα σφάλματα μοντελοποίσης αποδεικνύουν την υπερτερότητα της μοντελοποίησης που παρέχει το νευρωνικό δίκτυο. Για μια ακριβέστερη εικόνα παραθέτουμε γραφικά και τις τιμές που προκύπτουν από τα μοντέλα σε σύγκριση με τις πραγματικές τιμές που μας παρείχαν οι εκτελέσεις στα Σχήματα 5.16,5.17 και 5.18 σε σχέση με μία μόνο μεταβαλλόμενη παράμετρο κάθε φορά.




*Σχήμα 5.17: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο πλήθος instances*




*Σχήμα 5.18: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο πλήθος dimensions*




*Σχήμα 5.19: Μοντελοποίηση χρονικής εκτέλεσης για μεταβαλλόμενο πλήθος cluster*

Αυτό που παρατηρούμε από τα παραπάνω γραφήματα για τις συγκεκριμένες περιπτώσεις είναι αυτό που αναμέναμε από τις τιμές των σφαλμάτων μοντελοποίησης. Δηλαδή το νευρωνικό δίκτυο σαφώς υπερτερεί του tree model και ειδικότερα στη περίπτωση της εναλλαγής του πλήθους των instances παρατηρούμε και μετρήσεις που η διαφορά ακρίβειας μεταξύ των δύο μοντέλων γίνεται μεγάλη. Σε γενικές γραμμές όμως και τα δύο μοντέλα παρεμβάλλονται ακρούντως καλά στις πραγματικές τιμές τις οποίες πήραμε και μπορούν να θεωρηθούν αξιόπιστα και χρήσιμα.

Το αντίβαρο στην ακρίβεια που προσφέρει το νευρωνικό δίκτυο σε σχέση με το tree model είναι ότι το τελευταίο, ειδικά για αυτή τη περίπτωση είναι ιδιαίτερα απλό. Συγκεκριμένα με τη δυνατότητα που μας παρέχει το weka να οπτικοποιήσουμε τη δεντρική του δομή παρατηρήσαμε ότι αυτό έχει μέγιστο βάθος δυο επιπέδων και στηρίζεται μόνο σε διαχωρισμούς των τιμών του cluster και των dimensions για να καταλήξει στα γραμμικά μοντέλα των φύλλων του. Η απλότητα αυτή που παρατηρήσαμε σε συνδυασμό με την αρκετά καλή προσέγγιση που βλέπουμε ότι αυτό προσφέρει ήδη, μας οδηγεί στο συμπέρασμα ότι στη πιθανή τροφοδότηση του μοντέλου με ακόμα περισσότερες τιμές εκτελέσεων θα είναι αντίστοιχα πολύ ακριβές με το νευρωνικό δίκτυο.

### 5.4.3 Μοντελοποίηση χρήσης μνήμης

Για την μοντελοποίηση χρήσης μνήμης παραθέτουμε στο Πίνακ 5.7, όπως και στα προηγούμενα, ένα δείγμα των δεδομένων μοντελοποίησης που χρησιμοποιήσαμε για την κατασκευή των μοντέλων μας.

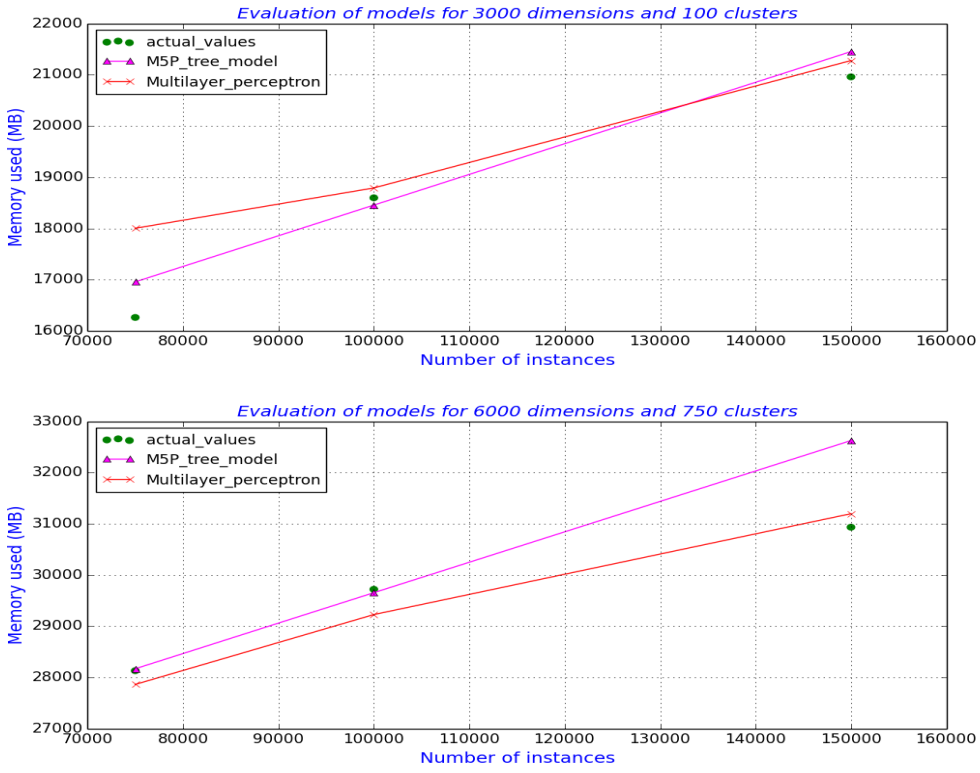
| Instances | Dimensions | Clusters | Memory used (MB) |
|---|---|---|---|
| 75000 | 3000 | 100 | 16261 |
| 100000 | 4000 | 500 | 25812 |
| 150000 | 6000 | 750 | 30932 |

*Πίνακας 5.7: Δείγμα δεδομένων μοντελοποίησης χρησιμοποιούμενης μνήμης*

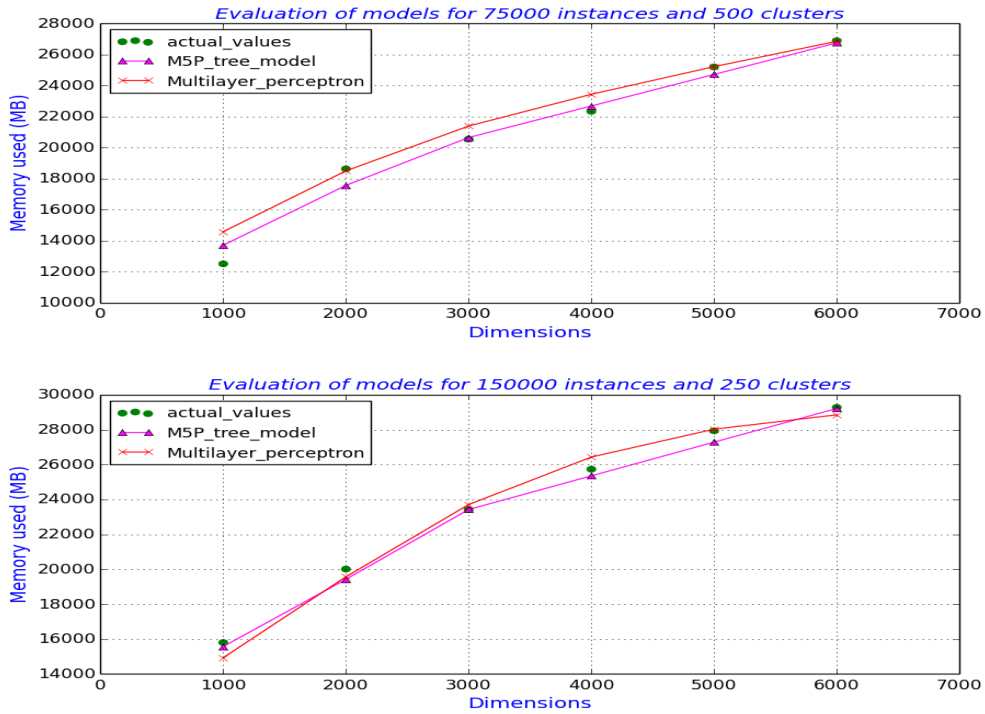
Τα μοντέλα που χρησιμοποιήσαμε ήταν και πάλι το Multilayer Perceptron και το M5P tree model. Τα σφάλματα που προέκυψαν από τη διαδικασία επαλήθευσής τους είναι αυτά που φαίνονται στον Πίνακα 5.6 παρακάτω ενώ στα Σχήματα 5.18,5.19,5.20 βλέπουμε γραφικά τα αποτελέσματα της μοντελοποίησης.

| Models | Mean Absolute Error (MB) | Mean Relative Error |
|---|---|---|
| M5P tree model | 1189 | 31.51% |
| Multilayer Perceptron | 646 | 17.12% |

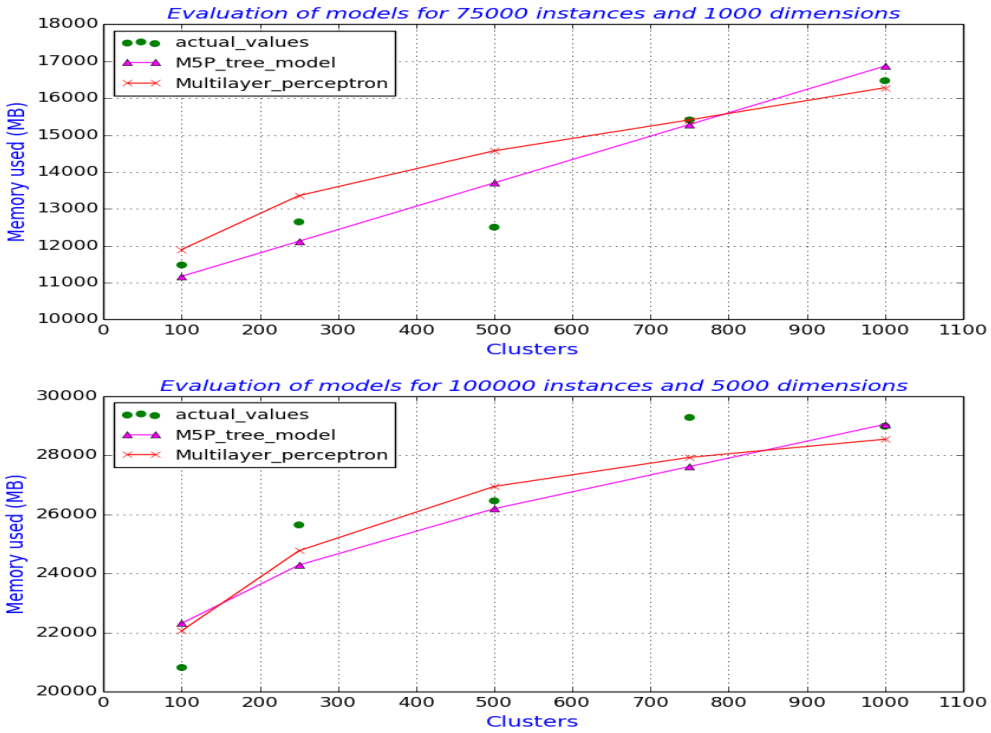
*Πίνακας 5.8: Σφάλματα μοντελοποίησης χρησιμοποιούμενης μνήμης*




*Σχήμα 5.20: Μοντελοποίηση χρησιμοποιούμενης μνήμης για μεταβαλλόμενο πλήθος instances*




*Σχήμα 5.21: Μοντελοποίηση χρησιμοποιούμενης μνήμης για μεταβαλλόμενο πλήθος dimensions*




*Σχήμα 5.22: Μοντελοποίηση χρησιμοποιούμενης μνήμης για μεταβαλλόμενο πλήθος cluster*

Από τα παραπάνω σχήματα βλέπουμε ότι και τα δύο μοντέλα πετυχαίνουν πολύ καλή ακρίβεια στη μοντελοποίηση της χρησιμοποιούμενης μνήμης. Ειδικότερα για το tree model παρατηρούμε ακόμα μεγαλύτερη ακρίβεια σε μερικές περιπτώσεις όπως προκύπτουν από τις

καμπύλες για μεταβαλλόμενο πλήθος instances. Ισχύει επίσης για το tree model αυτό που αναφέραμε και στη προηγούμενη ενότητα για την μοντελοποίηση του χρόνου εκτέλεσης, ότι η δενδρική δομή του μοντέλου παραμένει και σε αυτή τη περίπτωση πολύ απλή και σε συνδυασμό με την εκφραστική του ικανότητα το κάνει μία πολύ καλή και ελκυστική λύση μοντελοποίησης

Όπως έχουμε δει η χρησιμοποιούμενη μνήμη μετά από κάποιες τιμές των παραμέτρων που ορίζουν μεγάλα dataset όπως αυτά για τα 6000 dimensions και 750 clusters στο Σχήμα 5.19 αγγίζει τον κορεσμό ενώ από εκεί και πέρα έχουμε προβλήματα στις εκτελέσεις που σχετίζονται με τη μνήμη. Τα παραπάνω μεταφράζονται στο ότι η αξία των παραπάνω μοντέλων είναι διττή διότι εκτός του γεγονότος ότι μπορούν δυνητικά να μας εκφράσουν το ποσοστό μνήμης που χρειάζεται μία εκτέλεση μπορούν επίσης να μας δώσουν και μια εικόνα των ορίων των dataset που μπορούμε να χρησιμοποιήσουμε αναμένοντας ομαλή εκτέλεση.

# Κεφάλαιο 6

# Επίλογος

## 6.1 Σύνοψη και συμπεράσματα

Στην εργασία αυτή μελετήσαμε και μοντελοποιήσαμε την εκτέλεση του αλγορίθμου συσταδοποίησης k-means κεντρικά στο Weka και κατανεμημένα στο Apache Spark. Για το σκοπό αυτό προτείναμε και κατασκευάσαμε δύο αρχιτεκτονικές profiling με τις οποίες συλλέξαμε τις μετρικές επίδοσης που σχετίζονταν με τους χρησιμοποιούμενους υπολογιστικούς πόρους για κάθε εκτέλεση και σε κάθε μηχανή και επίσης την χρονική επίδοση, για μεταβαλλόμενες τιμές πλήθους instances,dimensions, clusters. Με τις μετρήσεις αυτές καταφέραμε για κάθε περίπτωση μετρικής και μηχανής εκτέλεσης να κατασκευάσουμε μοντέλα που προσομοιώνουν ικανοποιητικά την κάθε μετρική για κάθε πιθανή μεταβολή των παραμέτρων εκτέλεσης του αλγορίθμου.

Για την υλοποίηση σε Weka κατασκευάσαμε ένα σύνολο datasets με διαφορετικούς αριθμούς instances και dimensions ανά instance και εξακριβώσαμε αρχικά την σταθερή χρονική εξάρτηση των εκτελέσεων από το πλήθος των επαναλήψεων καθώς και τον περιορισμό του Weka με τη χρήση μόνο ενός core ανά μηχάνημα. Αποδείξαμε την πλήρη αντιστοίχιση κατά μέγεθος του disk input με το υποκείμενο dataset και την σχεδόν μηδενική χρήση του δίσκου για εγγραφές. Παρατηρήσαμε τη γραμμική εξάρτηση τόσο της χρονικής επίδοσης καθώς και τη χρήσης μνήμης καθώς μεταβάλλουμε κάθε μία από τις παραμέτρους. Σχετικά με τη χρονική επίδοση αναφέραμε συνολικά, ότι εντοπίζουμε μη ενθαρρυντικές τιμές κυρίως ως απόρροια της μη παράλληλης εκτέλεσης του αλγορίθμου σε περισσότερα cores ενώ αντίστοιχα αποθαρρυντικές ήταν και οι μετρήσεις μας σχετικά με τη χρήση μνήμης όπου παρατηρήθηκαν δυσανάλογα υψηλές τιμές σε σχέση με το μέγεθος του dataset.

Αντίστοιχη πορεία μελέτης ακολουθήσαμε και για την εξέταση της κατανεμημένης υλοποίησης στο Apache Spark. Σε αυτή τη περίπτωση επίσης εντοπίσαμε σταθερή χρονική εξάρτηση από το πλήθος των επαναλήψεων που εκτελεί ο αλγόριθμος. Για την μέτρηση disk input παρατηρήσαμε διαχωρίσιμες συμπεριφορές για dataset που είναι μικρότερα της συνολικής cache του cluster και για dataset που είναι μεγαλύτερα. Για τη μέτρηση του disk

output παρατηρήσαμε ταυτόχρονη εξάρτηση τόσο από το μέγεθος του dataset αλλά και από το μέγεθος των shuffle write δεδομένων στο τέλος κάθε επανάληψης. Αυτό μεταφράστηκε σε παραβολικού τύπου αύξηση της μέτρησης καθώς αυξάναμε τις παραμέτρους των cluster και των dimensions. Είδαμε γραμμμική χρονική εξάρτηση για μεταβολές των παραμέτρων, ενώ επισημάναμε σε ποιο στάδιο της εκτέλεσης προσθέτει η κάθε παράμετρος χρονική επιβέρυνση. Παράλληλα εξετάσαμε την χρήση της μνήμης από όλο το cluster και τα όρια κορεσμού της μετά από συγκεκριμένα μεγέθη dataset. Στο σημείο εκείνο αναφέραμε και τη παρατήρηση που κάναμε σχετικά με την αδυναμία του Spark να αναλάβει εκτελέσεις του αλγορίθμου με dataset που ξεπερνούν συγκεκριμένα μεγέθη με σφάλματα που μας παρέπεμψαν σε προβλήματα διαθέσιμης μνήμης.

Επιχειρήσαμε επίσης σύγκριση των υλοποιήσεων Weka και Spark. Αποδείχθηκε η ανωτερότητα της κατανεμημένης εκδοχής ακόμα ίσως και για μεγέθη dataset που θεωρητικώς θα έπρεπε να υστερεί της αντίστοιχης κεντρικής. Το γεγονός αυτό μας ανέδειξε την πολύ καλή σχεδίαση του Spark σχετικά με την αρχιτεκτονική εκτέλεσης που αυτό υιοθετεί. Επίσης ελέγξαμε την κλιμακωσιμότητα του αλγορίθμου στο Spark πραγματοποιώντας τις ίδιες εκτελέσεις σε cluster με διαφορετικό αριθμό nodes. Τα αποτελέσματα δείξανε σημαντική χρονική βελτίωση της τάξης του 25% κατά μέσο όρό.

Τέλος για όλες τις παραπάνω μετρικές κατασκευάσαμε ακριβή μοντέλα που προσομοιώνουν την συμπεριφορά τους. Πειραματιστήκαμε με ένα σύνολο διαθέσιμων μοντέλων που προσφέρει το Weka και από αυτά χρησιμοποιήσαμε γραμμικά μοντέλα, νευρωνικά δίκτυα και M5P δέντρα απόφασης. Κατά περίπτωση μετρικής και μηχανής εκτέλεσης αποδείξαμε την ακρίβεια των παραπάνω. Το τελευταίο ήταν το επιστέγασμα της ολοκλήρωσης του σκοπού που είχαμε αρχικώς θέσει, δηλαδή η κατασκευή μοντέλων που να εκφράζουν συνολικά την εκτέλεση του αλγορίθμου k-means σε αυτές τις δυο μηχανές.

## 6.2 Μελλοντικές Επεκτάσεις

Το Weka και το Apache Spark είναι μόνο δύο από τις πιθανές μηχανές εκτέλεσης που η παρούσα εργασία ασχολήθηκε. Ως συνέχεια της μελέτης που ξεκινήσαμε προτείνουμε αντίστοιχη πορεία μελέτης και για άλλες μηχανές όπως για παράδειγμα η υλοποίηση του k-means με το Apache Mahout στο Hadoop. Η τελευταία είναι ακόμα μία κατανεμημένη επιλογή η οποία θα παρέχει ακόμα πιο χρήσιμες συγκρίσεις τουλάχιστον όσον αφορά με το Apache Spark που ήδη είδαμε.

Ταυτόχρονα στη παρούσα εργασία περιοριστήκαμε σε δύο μόνο διαφορετικά μεγέθη cluster με συγκεκριμένο τύπο αριθμό cores ανά node. Η εναλλαγή των παραπάνω τιμών προφανώς επηρεάζει ποικιλοτρόπως την εκτέλεση του αλγορίθμου και τις αντίστοιχες μετρήσεις. Μεγάλο ενδιαφέρον θα παρουσίαζε ο πειραματισμός με πολλές διαφορετικές τιμές των ανωτέρω και η εμπλούτιση των δεδομένων μοντελοποίησης με τις τιμές αυτές. Με τον τρόπο αυτό θα μπορούσαμε να κατασκευάσουμε καθολικά μοντέλα για κάθε μηχανή για κάθε πιθανό τρόπο εκτέλεσης τόσο όσο αναφορά τις παραμέτρους του αλγορίθμου αλλά και τις παραμέτρους που ορίζουν την φυσική υπόσταση του cluster.

# Βιβλιογραφία


[1] D.Clark,"Forbes",http://www.forbes.com/sites/dorieclark/2013/08/08/four-things-you-need-to-know-in-the-big-data-era/.

[2] "Amazon Web Services" Amazon, http://aws.amazon.com/.

[3] "Microsoft Azure" Microsoft, http://azure.microsoft.com/el-gr/.

[4] C. Mantas, D.Tsoumakos, "The Case for Multi-Engine Data Analytics," in *Euro-Par*, Aachen, 2013.

[5] N. Papailiou, D. Tsoumakos, C. Mantas, N. Koziris, K. Doka, "IReS: Intelligent, Mulit-Engine REsource Scheduler for Big Data Analytics Workflows," in *ACM SIGMOD/PODS International Conference on Management of Data*, Melbourne, 2015.

[6] Srinivasa Rao Pathakota, T. M. Srinivasa and Madhu Yedla, "Enhancing K-means Clustering Algorithm with Improved Initial Center," *International Journal of Computer Science and Information Technologies,* vol. 1, 2010.

[7] Wikipedia, "k-means clustering", https://en.wikipedia.org/wiki/K-means_clustering.

[8] S. Ghemawat, J.Dean, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004.

[9] H. Gobioff, Shun-Tak Leung and S. Ghemawat, "The Google File System," in *SOSP'03 Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[10] "Scala Programming Language", http://www.scala-lang.org/.

[11] "Apache Spark", https://spark.apache.org/.

[12] "Weka", http://www.cs.waikato.ac.nz/ml/weka/.

[13] Wikipedia,"LinearRegression",https://en.wikipedia.org/wiki/Linear_regression.

[14] Eibe Frank, Marl A. Hall and Ian H.Witten, Data Mining, Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2011.

[15] Wikipedia,"Backpropagation", https://en.wikipedia.org/wiki/Backpropagation.

[16] "Ganglia Monitoring System", http://ganglia.sourceforge.net/.

[17] Brent N. Chun, David E. Culler and Mathhew L. Massie, "The ganglia distributed monitoring system: design implementation and experience," *Paraller Computing,* vol. 30, 2004.

[18] "Apache Web Server" , http://httpd.apache.org/.

[19] Bernard Li, Brad Nicholes, Vladimir Vuskan and Matt Massie, Monitoring with Ganglia, O'Reilly Media, 2012.

[20] "Iostat man page", http://linux.die.net/man/1/iostat.

[21] "Openstack", https://www.openstack.org/.

[22] D. Tsoumakos, N. Papailiou, I. Giannakopoulos, N. Koziris "PANIC: Modeling Application Performance over Virtualized Resources".