Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

Μοντελοποίηση της Επίδοσης και του Κόστους Αλγορίθμων Συνένωσης σε Εφαρμογές Μεγάλου Όγκου Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΓΙΩΡΓΟΣ ΔΑΜΑΣΚΗΝΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

Μοντελοποίηση της Επίδοσης και του Κόστους Αλγορίθμων Συνένωσης σε Εφαρμογές Μεγάλου Όγκου Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΓΙΩΡΓΟΣ ΔΑΜΑΣΚΗΝΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Ιουλίου 2015.

Νεκτάριος Κοζύρης	Δημήτριος Τσουμάκος	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ιονίου Πανεπιστημίου	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

Γιώργος Δαμασκηνός

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η συνεχής εξέλιξη της επιστήμης των υπολογιστών έχει επιφέρει ραγδαία αύξηση στον όγκο των δεδομένων που αποθηκεύονται συνεχώς σε διάφορα data - center. Είναι γεγονός ότι τα δεδομένα αυτά περιέχουν απίστευτα χρήσιμη πληροφορία. Μία άκρως ενδιαφέρουσα και πολυπληθής συλλογή από μηχανές, βιβλιοθήκες και τεχνικές είναι διαθέσιμη για κάθε χρήστη που προσπαθεί να εξαγάγει αυτήν την πληροφορία προκειμένου να καταλήξει σε ενδιαφέροντα συμπεράσματα. Ωστόσο λόγω του μεγέθους και της πολυπλοκότητας αυτής της συλλογής είναι πρακτικά αδύνατο για έναν μέσο χρήστη να επιλέξει τον ιδανικό συνδυασμό, βασισμένος στον σκοπό και στους διαθέσιμους πόρους του.

Σκοπός της παρούσας διπλωματικής είναι η συνεισφορά στον σχεδιασμό και στην υλοποίηση ενός συστήματος που θα δέχεται ως είσοδο από έναν χρήστη, ή ένα άλλο σύστημα, τους διαθέσιμους πόρους, το είδος της εργασία, καθώς και το επιθυμητό αποτέλεσμα στα πλαίσια βελτιστοποίησης κάποιας παραμέτρου επίδοσης (π.χ. ελάχιστος χρόνος εκτέλεσης). Στη συνέχεια, βασιζόμενο στα κατάλληλα μοντέλα, θα προτείνει στον χρήστη τον ιδανικό τρόπο εκτέλεσης της εργασίας του. Το είδος της εργασίας του χρήστη στην εν λόγω μελέτη, περιορίζεται σε αλγορίθμους συνένωσης μεγάλου όγκου δεδομένων.

Η επίτευξη του σκοπού αυτού γίνεται διαμέσου της υλοποίησης διαφόρων αλγορίθμων συνένωσης σε διάφορες μηχανές. Στην συνέχεια παρακολουθείται το προφίλ της επίδοσης και του κόστους αυτών των αλγορίθμων για διάφορους συνδυασμούς παραμέτρων. Τέλος δημιουργούνται τα κατάλληλα μοντέλα τα οποία το σύστημα θα εξετάζει προκειμένου να πετύχει το ζητούμενο.

Η διπλωματική αυτή, αποτελεί τμήμα μίας πλατφόρμας προσαρμοστικής και κλιμακώσιμης ανάλυσης δεδομένων μεγάλου όγκου (ASAP) και πιο συγκεκριμένα ενός ευφυούς δρομολογητή (IReS), ο οποίος είναι υπεύθυνος για την έξυπνη διαχείριση των υπαρχόντων πόρων.

Λέξεις κλειδιά

μεγάλου όγκου δεδομένα, εξόρυξη δεδομένων, πρόβλεψη επίδοσης, μοντελοποίηση, αλγόριθμοι συνένωσης, ASAP, IReS

Abstract

The continuous development of computer science, has led to a rapid increase in the amount of data that are constantly being stored at various data centers around the world. It is a fact, that this data contains tremendously valuable information. A most interesting assortment of engines, libraries and techniques for big data analytics is available for every user who wish to extract this valuable information in order to reach some potentially groundbreaking results. Nevertheless, due to the size and complexity of this assortment, it is practically impossible for a non expert user to identify the optimal combination, based on his goal and resources.

The purpose of this thesis is to contibute to the design and implementation a system that takes as input from a user, or another system, the available resources, the type of the application and the desired result, as far as optimizing various performance metrics is concerned (e.g. minimize execution time). Based on this input, it will be able to derive, according to a collection of models, the optimal way of executing the particular job. This study is focused into join algorithms as far as the application is concerned.

This goal is achieved by implementing various join algorithms for various execution engines. The profiling of the cost and performance of these join algorithms for different configurations, is what follows. Finally, taking the results into consideration, the appropriate surrogate models are created in order for the system to be able to achieve it's functionality.

This thesis is part of an adaptive and scalable analytics platform, ASAP. In more detail, it is a part of an intelligent resource scheduler (IReS) that provides the ASAP with intelligent resource management.

Key words

bid data analytics, data mining, performance prediction, modeling, profiling, surrogate models, join algorithms, ASAP, IReS

Ευχαριστίες

Η παρούσα διπλωματική ολοκληρώνει την 5ετή φοίτηση μου στην σχολή των Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Νεκτάριο Κοζύρη για την εμπιστοσύνη και την υποστήριξη που έδειξε στα πλαίσια εκπόνησης αυτής της διπλωματικής, αλλά επίσης και για την έμπνευση και το ενδιαφέρον που μου προκάλεσε διαμέσου της διδασκαλίας του και των προπτυχιακών μαθημάτων του. Ήταν ένας από τους λόγους που αποφάσισα να ασχοληθώ με τον τομέα των υπολογιστικών συστημάτων γενικά, και των κατανεμημένων συστημάτων και την ανάλυση δεδομένων μεγάλου όγκου ειδικά.

Ταυτόχρονα οφείλω ένα μεγάλο ευχαριστώ στην Δρα. Κατερίνα Δόκα για την συνεχή υποστήριξη και καθοδήγηση που οδήγησε στην επιτυχή έκβαση της μου διπλωματικής εργασίας. Το ενδιαφέρον της και οι συζητήσεις μας υλοποίησαν την αρχική ιδέα με επιτυχία και έδωσαν λύση στα διάφορα προβλήματα που προέκυψαν στην πορία.

Επιπρόσθετα θα ήταν σημαντική παράλειψη να μην ευχαριστήσω τον Υ.Δ Γιάννη Γιαννακόπουλο καθώς και τον Δημήτρη Σαρλή για την καθοριστική συμβολή τους σε σημαντικά τεχνικά προβλήματα που αντιμετωπίστηκαν κατά τη διάρκεια αυτής της διπλωματικής.

Τέλος, όπως σε κάθε σημαντική και δύσκολη φάση της ζωής του καθενός, η οικογένεια μου και οι φίλοι μου με βοήθησαν σημαντικά στον ψυχολογικό παράγοντα. Χωρίς ψυχικό - συναισθηματική υγεία και σταθερότητα, τίποτα δεν είναι εφικτό.

Γιώργος Δαμασκηνός,

Αθήνα, 14η Ιουλίου 2015

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κατάλογος Κώδικα

Κεφάλαιο 1

Εισαγωγή

Η ανάπτυξη του διαδικτύου και η εδραίωσή του ως αποτελεσματικού τρόπου διασύνδεσης δισεκατομμυρίων χρηστών - μηχανών σε συνδυασμό με την τρομακτική ανάπτυξη της τεχνολογίας των υπολογιστικών συστημάτων, οδήγησε στην δημιουργία νέων περιοχών απαράμιλλου ενδιαφέροντος και σημασίας στην επιστήμη των υπολογιστών. Ένας από τους πιο τελευταία αναπτυσσόμενους αυτούς τομείς είναι αυτός της ανάλυσης δεδομένων μεγάλου όγκου (Big Data Analytics).

Στον τομέα αυτό, επίκεντρο είναι τα δεδομένα τα οποία μπορούν να χαρακτηριστούν από τα 3Vs:

1. Τεράστιος όγκος (extreme **Volume**)
2. Ποικιλομορφία (wide **Varierty**)
3. Τρόπος παροχής (**Velocity**)

Οι πηγές των δεδομένων αυτών είναι εν γένη εντελώς ασυσχέτιστες μεταξύ τους (εξού και η ποικιλομορφία στην δομή). Κάποιες βασικές κατηγορίες πηγών είναι: έγγραφα υπηρεσιών, εικονικά μέσα (media), επιχειρησιακές εφαρμογές, κοινωνικά δίκτυα, καταγραφές μηχανών (logs), αισθητήρες. Στο Σχ. 1.1 δίνεται μια εικόνα για τον τρόπο που δομούνται τα δεδομένα σε κάθε μία από αυτές τις πηγές.

Σχήμα 1.1: Πηγές Δεδομένων Μεγάλου Όγκου

Είναι εύκολο συνεπώς να παρατηρήσει κανείς την επιτακτικότητα της ανάλυσης αυτών των δεδομένων (analytics), και της εξόρυξης πολύτιμης πληροφορίας εξ' αυτών (data mining). Το πρόβλημα το οποίο κάνει τον νέο αυτό κλάδο τόσο σημαντικό και συνεχώς αναπτυσσόμενο, είναι ότι η υπάρχουσα τεχνολογία σε λογισμικό και υλικό δεν μπορεί να αντεπεξέλθει στον όγκο και στην ταχύτητα εισροής των δεδομένων αυτών. Τα "παραδοσιακά" εργαλεία ανάλυσης δεν επαρκούν για τόσο πολύπλοκα και δυναμικά δεδομένα. Επομένως είναι απαραίτητη η δημιουργία νέων εργαλείων, κατανεμημένων συστημάτων και τεχνικών.

Την ίδια περίοδο που ο τομέας αυτός γίνεται ολοένα και μεγαλύτερης σημασίας, τα υπολογιστικά συστήματα παρουσιάζουν αξιοθαύμαστη πρόοδο και εντυπωσιακά αποτελέσματα. Στο πλαίσιο αυτό, τα υπολογιστικά συστήματα νέφους (cloud computing systems) δίνουν νέες δυνατότητες σε τρεις βασικές κατηγορίες:

- Υποδομή ως υπηρεσία (Infrastructure-as-a-Service: IaaS)
- Πλατφόρμα ως υπηρεσία (Platform-as-a-Service: PaaS)
- Λογισμικό ως υπηρεσία (Software-as-a-Service: SaaS)

Ο συνδυασμός των υπηρεσιών αυτών με την τεχνογνωσία των υπολογιστικών συστημάτων, έχει ήδη οδηγήσει στην δημιουργία νέων μηχανών και εργαλείων, τα οποία έχουν καταστήσει δυνατή την αντιμετώπιση και ανάλυση δεδομένων μεγάλου όγκου. Απώτερος σκοπός παραμένει ωστόσο η επίδοση και η αποτελεσματικότητα.

Οι πολύτιμοι και πεπερασμένοι υπολογιστικοί πόροι, που παρότι ολοένα και αυξάνονται, έχουν πολύπλοκα όρια στις επιδόσεις του ανάλογα με τον τρόπο δόμησής τους. Στόχος παραμένει η αποδοτικότερη από πλευράς κόστους (έννοια σχετική για κάθε περίπτωση) χρησιμοποίηση των πόρων αυτών.

Τα ήδη πολλά υποσχόμενα αποτελέσματα, αναμένουν ακόμα μεγαλύτερη πρόοδο, γεγονός που θα συντελέσει στην σαφή βελτίωση της οικονομικής, κοινωνικής και πολιτικής ζωής των ανθρώπων. Εξάλλου η τεχνολογίες αυτές, όπως και κάθε τεχνολογία υπάρχουν και θα πρέπει να υπάρχουν και να αναπτύσσονται σε ένα ανθρωποκεντρικού χαρακτήρα πλαίσιο.

1.1 Κίνητρο της Διπλωματικής Εργασίας

Τα εργαλεία και τα συστήματα που έχουν αναπτυχθεί και αναπτύσσονται, είναι ήδη αρκετά πολύπλοκα στην χρήση για κάποιο μη - ειδικό χρήστη. Ακόμα όμως και για έναν ειδικό, η συνεχής αύξησή τους σε αριθμό και πολυπλοκότητα, καθιστά πρακτικά αδύνατη την εξονυχιστική εξέταση της επίδοσης όλων, προκειμένου να βρεθεί το ιδανικό για το συγκεκριμένο προφίλ και δουλειά του χρήστη. Εξάλλου, στα εργαλεία αυτά, δεν είναι δεδομένο το σύνολο των εφαρμογών στα οποία υπερισχύουν έναντι των άλλων. Επίσης διαθέτουν πληθώρα ρυθμίσεων οι οποίες επηρεάζουν καθοριστικά την επίδοσή τους.

Δεδομένου ότι υπάρχουν πολλές μηχανές και βιβλιοθήκες που στοχεύουν στην βελτιστοποίηση συγκεκριμένων εργασιών και συγκεκριμένων τύπων δεδομένων, στην περίοδο αυτή γίνεται λόγος για multi-engine περιβάλλοντα (π.χ. εικονικές μηχανές με προεγκατεστημένες πολλές μηχανές). Για να είναι σε θέση ένας χρήστης να επιλέξει την κατάλληλη για την εργασία του μηχανή καθώς επίσης και με ποια συγκεκριμένη υλοποίηση θα την εκτελέσει, είναι απαραίτητη κάποιου είδους πληροφορία πρόβλεψης.

Προς αυτή την κατεύθυνση βρίσκεται και ο στόχος της παρούσας διπλωματικής. Πιο συγκεκριμένα, είναι η δημιουργία ενός ***πλαισίου μοντελοποίησης*** που θα περιγράφει λεπτομερώς την επίδοση και το κόστος συγκεκριμένων υλοποιήσεων αλγορίθμων πάνω σε συγκεκριμένες μηχανές εκτέλεσης λαμβάνοντας υπόψιν πληθώρα παραμέτρων. Οι παράμετροι αυτές μπορεί να σχετίζονται με τα δεδομένα εισόδου (π.χ. μέγεθος, κατανομή κλπ), με τον ίδιο τον αλγόριθμο ή με την υποδομή (π.χ. αριθμός κόμβων κλπ.).

Σκοπός του πλαισίου αυτού είναι η επιλογή του συνδυασμού υλοποίησης ενός αλγορίθμου και μηχανής που βελτιστοποιεί συγκεκριμένα κριτήρια, π.χ. χρόνο εκτέλεσης ή χρησιμοποίηση CPU κλπ. Το περιεχόμενο και ο σκοπός αυτός, είναι πλήρως ευθυγραμμισμένος με έναν από τους βασικούς σκοπούς που επιδιώκει το ASAP [1] το οποίο είναι ένα execution framework για big data analytics.

Σε αντίθεση με τα [11], [5] σκοπός δεν είναι ο διαχωρισμός του font-end workflow από το back-end execution στις διαθέσιμες μηχανές, αλλά η ενιαία συμπεριφορά της εφαρμογής για κάθε μηχανή εκτέλεσης. Παρεμφερή περιεχόμενο και σκοπό έχει και το σύστημα PANIC [10], στόχος του οποίου είναι η πρόβλεψη της επίδοσης για πολύπλοκες εφαρμογές εκτελούμενες σε υπολογιστικό νέφος. Η επίτευξη του πραγματοποιείται μέσω της πραγματικής εκτέλεσης της εφαρμογής σε αντίθεση με την εν λόγω διπλωματική, όπου το πλαίσιο μοντελοποίησης δεν απαιτεί την πραγματική εκτέλεση περαιτέρω εφαρμογών μετά την δημιουργία του.

1.2 Δομή της Διπλωματικής Εργασίας

Η δομή της διπλωματικής εργασίας είναι η ακόλουθη:

Κεφάλαιο 2
Σε αυτό το κεφάλαιο, παρουσιάζεται το απαραίτητο υπόβαθρο, τόσο θεωρητικό όσο και τεχνολογικό, ώστε να μπορέσει ο αναγνώστης να γνωρίσει τις βασικές αρχές και μηχανισμούς, που σχετίζονται με τη διπλωματική αυτή.

Κεφάλαιο 3
Σε αυτό το κεφάλαιο, παρουσιάζονται οι βασικές σχεδιαστικές αποφάσεις για την υλοποίηση του συστήματος. Παράλληλα γίνεται περιγραφή των τεχνικών λεπτομερειών που έπαιξαν καθοριστικό ρόλο.

Κεφάλαιο 4
Σε αυτό το κεφάλαιο, παρουσιάζεται η υλοποίηση του συστήματος, καθώς επίσης και ερμηνεία των αποτελεσμάτων που προέκυψαν.

Κεφάλαιο 5
Σε αυτό το κεφάλαιο κάποιες προτάσεις για μελλοντικές επεκτάσεις και βελτιώσεις του συστήματος, καθώς και κάποια γενικά σχόλια.

[1] http://www.asap-fp7.eu/

Κεφάλαιο 2

Θεωρητικό και Τεχνολογικό Υπόβαθρο

Σε αυτό το κεφάλαιο, παρουσιάζεται το απαραίτητο υπόβαθρο, τόσο θεωρητικό όσο και τεχνολογικό, ώστε να μπορέσει ο αναγνώστης να γνωρίσει τις βασικές αρχές και μηχανισμούς, που σχετίζονται με τη διπλωματική αυτή. Για κάθε εργαλείο και σύστημα που χρησιμοποιήθηκε, γίνεται μία σύντομη ανάλυση της βασικής του λειτουργίας, και μια αναφορά στον λόγο και τον σκοπό για τον οποίο επιλέχτηκε. Επιπροσθέτως, γίνεται μια σύντομη επεξήγηση της λειτουργίας των αλγορίθμων που χρησιμοποιήθηκαν.

Πιο συγκεκριμένα, στο Τμήμα 2.1 γίνεται μία σύντομη ανάλυση των αλγορίθμων συνένωσης και των βασικών χαρακτηριστικών τους. Στο Τμήμα 2.2 γίνεται αναφορά στην έννοια της μοντελοποίησης και στο εργαλείο μοντελοποίησης που επιλέχτηκε. Στο Τμήμα 2.3 περιγράφεται η ανάγκη για παρακολούθηση, και τα εργαλεία παρακολούθησης, σε επίπεδο μετρικών επίδοσης συστήματος του συμπλέγματος. Στο Τμήμα 2.4 δίνεται μια σύντομη αναφορά, στις μηχανές εκτέλεσης που χρησιμοποιήθηκαν και στους λόγους που αυτές επιλέχτηκαν. Ακόμα παρουσιάζονται τα βασικά στοιχεία από την κάθε μία που επιλέχτηκαν για τη μελέτη.

2.1 Αλγόριθμοι Συνένωσης

Η συνένωση (join) δύο ή περισσότερων πινάκων είναι μία πολύ συχνή και χρήσιμη λειτουργία σε εφαρμογές ΑΔΜΟ [19]. Ο τρόπος όμως που γίνεται στις σχεσιακές βάσεις δεδομένων είναι μη - κλιμακώσιμος και ασύμφορος μετά από κάποιο μέγεθος πινάκων και πάνω. Έτσι λοιπόν χρειάζεται η υλοποίησή των joins με τεχνικές ΑΔΜΟ όπως το MapReduce, και δυνατότητα εκτέλεσής τους στα διάφορα εργαλεία. Πιο συγκεκριμένα οι αλγόριθμοι αυτοί όπως υλοποιούνται στο MapReduce είναι: το Reduce-Side Join 2.1.1 και το Map-Side Join 2.1.2. Υποκατηγορία του τελευταίου είναι και το Broadcast Join 2.1.2.1 καί Bucket Join 2.1.2.2. [8]

Τα βασικά είδη joins, ως προς τον τρόπο που καθορίζεται ποιες θα είναι οι τελικές πλειάδες (tuples) του αποτελέσματος, που προβλέπει η standard SQL (inner, left outer, right outter, full outer, cross). Η συνθήκη που πρέπει να ικανοποιείται ώστε να γίνει η συνένωση δύο γραμμών του πίνακα είναι αυτή που καθορίζει το είδος των joins ως προς την συνθήκη (equi joins, theta joins κτλ). Οι αλγόριθμοι συνένωσης που πραγματεύεται η παρούσα διπλωματική, είναι αυτή της συνένωσης με βάση την ισότητα (equi joins), και περιορίζονται στην συνένωση δύο πινάκων.

2.1.1 Reduce - Side Join

Από τους υπάρχοντες αλγόριθμους, ο Reduce Side είναι ο πιο εύκολος όσο αφορά στην υλοποίηση. Ο κάθε Mapper αναλαμβάνει την επεξεργασία ενός τμήματος εκ των δύο πινάκων και παράγει ένα key-value ζευγάρι, όπου το value αποτελείται από το value της εισόδου μαζί με ένα αναγνωριστικό (tag) που δείχνει από ποιόν από τους δύο πίνακες προήλθε. Αφού γίνει το shuffle και οι εγγραφές με το ίδιο κλειδί, βρεθούνε στον ίδιο Reducer, ο τελευταίος αναλαμβάνει να ελέγξει αν υπάρχουν αναγνωριστικά και των δύο πινάκων. Στην περίπτωση αυτή, παράγει την ζητούμενη εγγραφή. Συνοπτικά τα παραπάνω παρουσιάζονται στο ακόλουθο τμήμα ψευδοκώδικα [20].

Algorithm 1 Reduce Side Join

```
1:  function Map (key,value)
2:      for each table i do
3:          for each record in table i do
4:              emit(key,(value,tag(i)))
5:          end for
6:      end for
7:  end function
8:
9:  function Reduce (key,list(value,tag(i)))
10:     for each key i do
11:         if tag1 and tag2 in list(value,tag(i)) then
12:             emit(key,list(value))
13:         end if
14:     end for
15: end function
```

Τα κύρια πλεονεκτήματα του Reduce Side Join είναι:

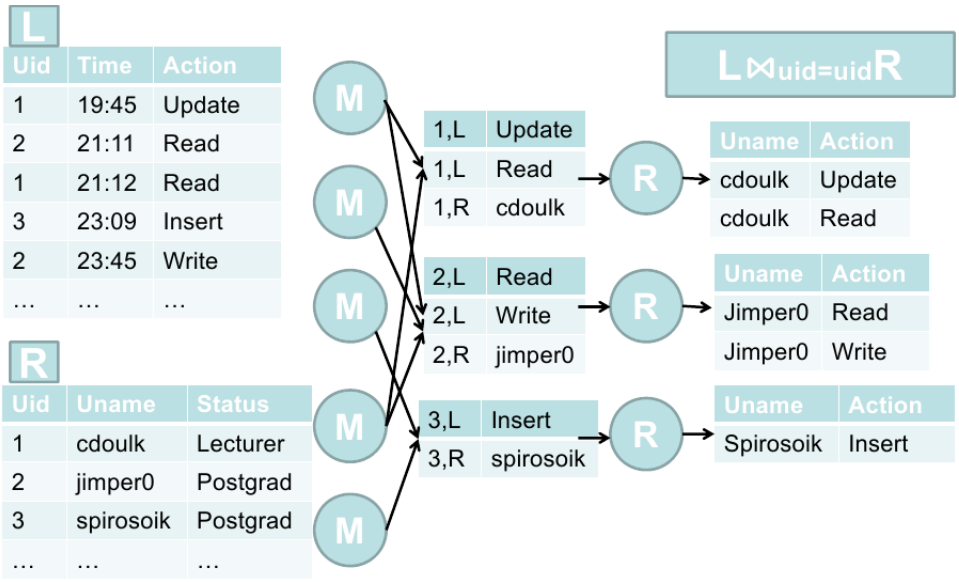
- Εύκολη υλοποίηση
- Μικρότερη (σχετικά με Map Side Join) χρησιμοποίηση μνήμης αφού μόνο οι πλειάδες με το ίδιο κλειδί χρειάζεται να αποθηκευτούν

Τα κύρια μειονεκτήματα του Reduce Side Join είναι:

- Κόστος για την μεταφορά δεδομένων στους Reducers που οφείλεται σε:
  - Shuffling
  - I/O και επικοινωνία
- Υψηλές απαιτήσεις μνήμης για σετ δεδομένων που παρουσιάζουν υψηλή πυκνότητα πλειάδων σε κάποια κλειδιά (skewed data)

2.1.2 Map - Side Join

Στην περίπτωση όπου κάθε είσοδος



Σχήμα 2.1: Παράδειγμα Reduce Side Join

- Είναι χωρισμένη στον ίδιο αριθμό διαμερίσεων
- Είναι ταξινομημένη με βάση το κλειδί της συνένωσης
- Έχει τον ίδιο αριθμό κλειδιών

μπορεί να εφαρμοστεί ο αλγόριθμος Map Side Join. Στον αλγόριθμο αυτό, η συνένωση πραγματοποιείται στην φάση map και ουσιαστικά δεν χρειάζεται η φάση της μεταφοράς των δεδομένων στους reducers.

Το κύριο πλεονέκτημα του Map Side Join είναι:

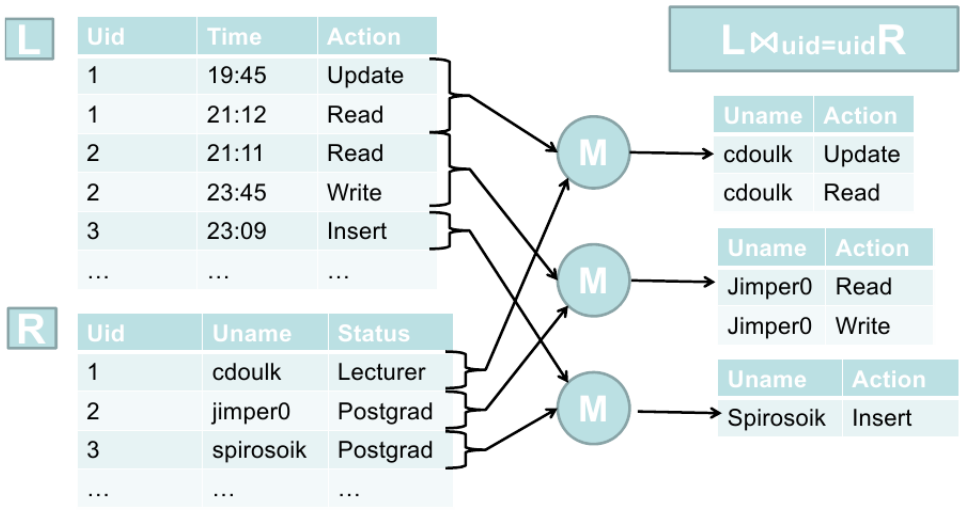
- Πολύ πιο αποδοτικό (σε σχέση με Reduce Side Join) αφού δεν υπάρχει κόστος για ενδιάμεσα δεδομένα και shuffling. Απλά γίνεται ανάγνωση των δεδομένων και συνένωσή τους.

Τα κύρια μειονεκτήματα του Map Side Join είναι:

- Αυστηρές απαιτήσεις για την είσοδο. Πρακτικά χρειάζεται άλλη μία MapReduce διαδικασία για να προ επεξεργαστεί κατάλληλα την είσοδο.
- Υψηλές απαιτήσεις κύριας μνήμης και οι δύο διαμερίσεις εισόδου αποθηκεύονται σε buffer μνήμη.

2.1.2.1 Broadcast Join

Στην ειδική περίπτωση, όπου ένας από τους δύο πίνακες είναι αρκετά μικρότερος από τον άλλο, μπορεί να χρησιμοποιηθεί ο αλγόριθμος του Broadcast Join, ο οποίος είναι υποκατηγορία του Map Side Join. Η λογική του είναι βασισμένη στο γεγονός ότι μπορεί να χρησιμοποιηθεί η κύρια μνήμη προκειμένου να αποθηκευτεί ο μικρός πίνακας και να αποσταλεί σε



Σχήμα 2.2: Παράδειγμα Map Side Join

κάθε διεργασία mapper. Έτσι λοιπόν οι υπολογισμοί που απαιτούνται για την παραγωγή του αποτελέσματος θα μειωθούν σημαντικά και θα υπάρξει βελτίωση της επίδοσης.

Ο mapper θα παίρνει στην μνήμη του όλο τον μικρό πίνακα, για τον οποίο και θα δημιουργεί ένα ευρετήριο κατακερματισμού (Hash Table). Στην συνέχεια θα δέχεται ένα τμήμα του μεγάλου πίνακα και για κάθε πλειάδα του, θα αναζητά στον πίνακα κατακερματισμού για αντίστοιχο κλειδί στον μικρό πίνακα. Είναι γεγονός ότι το κόστος αυτής της αναζήτησης λόγω του κατακερματισμού είναι σταθερό ($O(1)$). Αν υπάρχει αντίστοιχη τιμή, τότε το αντίστοιχο αποτέλεσμα θα παράγεται. Όπως φαίνεται και από το ακόλουθο τμήμα ψευδοκώδικα, ο αλγόριθμος αυτός δεν έχει reduce φάση.

Algorithm 2 Broadcast Join

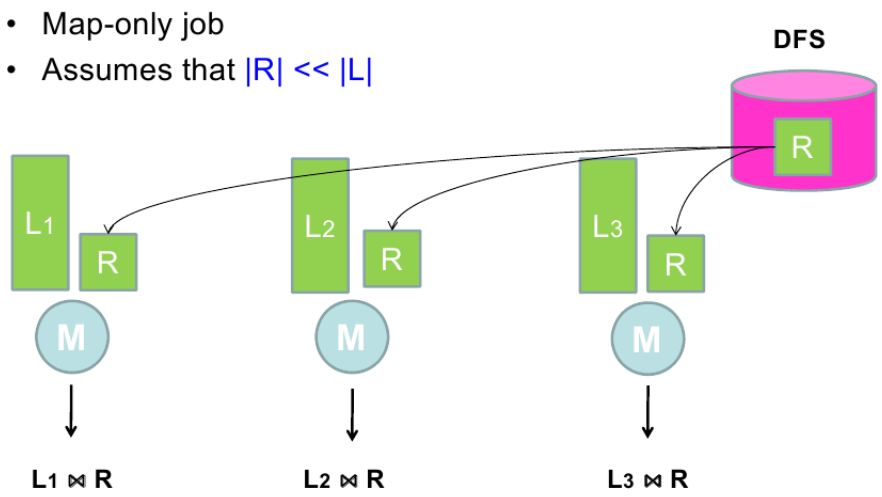
```
1: function Map (key,value)
2:     records = small_table.lookup(key)
3:     for rec : records do
4:         emit(value,rec)
5:     end for
6: end function
```

Το κύριο πλεονέκτημα του Broadcast Join επιπρόσθετο αυτών του Map Side είναι:

- Δεν χρειάζεται προ επεξεργασία των δεδομένων (π.χ. ταξινόμηση)

Το κύριο μειονεκτήματα του Map Side Join είναι:

- Το ένα από τα δύο σύνολα δεδομένων πρέπει να είναι αρκετά μικρό ώστε:
  - Να χωράει στην μνήμη
  - Να μπορεί να διανεμηθεί σε όλους τους mappers



Σχήμα 2.3: Παράδειγμα Broadcast Join

2.1.2.2 Bucket Join

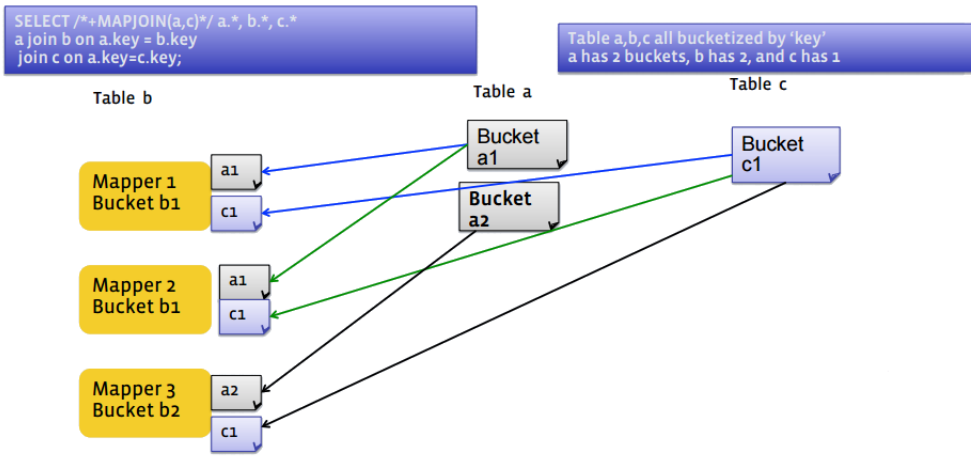
Το είδος αυτό join, όπως προαναφέρθηκε, ανήκει στην κατηγορία του Map Side Join. Στην περίπτωση όπου κανένας από τους πίνακες δεν είναι αρκετά μικρός ώστε να χωρέσει στην μνήμη, εφαρμόζεται η τεχνική bucket join. Στο bucket Join ο μικρός πίνακας χωρίζεται σε κάδους (buckets) με βάση το κλειδί. Οι κάδοι αυτοί θα πρέπει να είναι αρκετοί ώστε πλέον κάθε ένας από αυτούς να μπορεί να χωρέσει στην κύρια μνήμη. Στην συνέχεια ο κάθε κάδος αποθηκεύεται στον mapper, ώστε το κομμάτι του μεγάλου πίνακα που επίσης έχει αναλάβει ο εν λόγω mapper να αντιστοιχεί τα κλειδιά του, στα κλειδιά του bucket. Πιθανότατα κάποια buckets να αντιγραφούν σε διάφορους mappers ανάλογα με την πυκνότητα των κλειδιών.

Το κύριο πλεονέκτημα του Bucket Join είναι:

- Δίνει έναν τρόπο χρησιμοποίησης του Map Join (το οποίο όπως προαναφέρθηκε είναι αρκετά αποδοτικό), ακόμα και στην περίπτωση που κανένας από τους δύο πίνακες δεν χωράει στην μνήμη.

Το κύριο μειονεκτήματα του Map Side Join είναι:

- Επιπλέον κόστος στην λογική με την οποία καθορίζεται ένας πίνακας σε πόσους κάδους θα χωριστεί.



Σχήμα 2.4: Παράδειγμα Bucket Join

2.2 Μοντελοποίηση

Η μοντελοποίηση είναι μία πολύ σημαντική διαδικασία για διαφόρων ειδών εφαρμογές. Υπάρχουν πάρα πολλές περιπτώσεις οπού το φαινόμενο που εξετάζεται είναι τόσο περίπλοκα δομημένο, με πολλές εξαρτήσεις, που είναι πρακτικά αδύνατο κάποιος να προβλέψει ντετερμινιστικά τη συμπεριφορά του. Οι τεχνικές μοντελοποίησης δίνουν μία αξιόπιστη λύση με σχετικά μικρό κόστος.

Το cloud computing έχει ανοίξει νέους ορίζονται με τα πλεονεκτήματα και τις δυνατότητές του. Ωστόσο όμως, το virtualization, προσθέτει ένα επίπεδο λογισμικού πάνω από το υλικό, γεγονός που καθιστά την πρόβλεψη της αλληλεπίδρασης πόρων και επιδόσεων εφαρμογής αρκετά δύσκολη. Σε αυτή την κατεύθυνση τοποθετείται και η διαδικασία εξόρυξης δεδομένων, με επίκεντρο την επίδοση, και μοντελοποίησης τους για κάθε εφαρμογή [12] . Το Weka 2.2.1 είναι σχεδιασμένο ώστε ανάμεσα στα άλλα να επιτελεί και αυτό τον σκοπό.

2.2.1 Weka

Το Weka είναι μία συλλογή από αλγορίθμους machine learning για εξόρυξη δεδομένων σε εφαρμογές ΑΔΜΟ. Οι αλγόριθμοι μπορούν να εφαρμοστούν απευθείας στο σετ δεδομένων, ή να κληθούν από κάποιο κώδικα Java ως βιβλιοθήκες. Επίσης έχει αναπτυχθεί ένα πολύ εύχρηστο γραφικό περιβάλλον, που δίνει την δυνατότητα στον χρήστη να χρησιμοποιήσει τις δυνατότητες του Weka χωρίς την ανάγκη συγγραφής java κώδικα. Το Weka προσφέρει εργαλεία για :

- **Pre - processing tools**: Τα δεδομένα φορτώνονται με μορφή εισόδου csv κ.α. Στην συνέχεια επιλέγονται τα χαρακτηριστικά - στήλες (attributes) των δεδομένων αυτών που θα χρησιμοποιηθούν.
- **Classification tools**: Τα εργαλεία αυτά, δίνουν την δυνατότητα εκπαίδευσης κάποιου από τους υπάρχοντες ταξινομητές, με σκοπό την εξαγωγή μοντέλου. Για την εκπαίδευση, δίνονται διάφορες επιλογές που καθορίζουν το σύνολο εκπαίδευσης (training set),

και το σύνολο εξέτασης (test set). Το σύνολο εκπαίδευσης, είναι το σύνολο δεδομένων εισόδου, που θα πάρει ο ταξινομητής προκειμένου να βγάλει το μοντέλο. Το σύνολο εξέτασης, είναι το σύνολο δεδομένων που θα εφαρμοστεί αυτό το μοντέλο στην συνέχεια, ώστε να μπορέσει να εκτιμηθεί η απόδοσή του από πλευράς σφάλματος. Για αυτό και είναι γενικά σημαντικό να επιλέγονται σωστά και τα δύο αυτά σύνολα, ανάλογα με το είδος της εφαρμογής και των δεδομένων. Για παράδειγμα δεν θα ήταν συνετό να χρησιμοποιηθεί για σύνολο εκπαίδευσης το κομμάτι των δεδομένων που παρουσιάζει μία γραμμική τάση, και για για σύνολο εξέτασης ένα μετέπειτα κομμάτι που παρουσιάζει εκθετική, καθώς σε αυτή την περίπτωση τα σφάλματα θα ήταν δικαιολογημένα σημαντικά.

- **Visualization Tools**: Με αυτά τα εργαλεία, δίνεται η δυνατότητα γραφικής απεικόνισης των μοντέλων και των δεδομένων, καθώς επίσης και των σφαλμάτων. Έτσι ο χρήστης αποκτά μία σημαντική εποπτεία για την απόδοση των μοντέλων, η οποία είναι δύσκολο να εξαχθεί από την πληθώρα αποτελεσμάτων που παράγονται ειδικά για μεγάλα σετ δεδομένων. Συνεπώς, μπορεί να επιλέξει το ”βέλτιστο” κατά περίπτωση ταξινομητή και μοντέλο, εφαρμόζοντας μία τεχνική τύπου δοκιμής - λάθους (trial and error).

2.3 Συστήματα Παρακολούθησης

Τα αρχεία που περιέχουν λεπτομέρειες για διάφορες λειτουργίες του συστήματος (logs), περιέχουν πολύτιμο υλικό που αφορά τόσο πληροφορίες σχετικές με την πορεία και την επίδοση του συστήματος, όσο επίσης και πληροφορίες για πιθανά σφάλματα και αιτίες τους. Ταυτόχρονα, το κόστος διατήρησης αρχείων log είναι αμελητέο για το σύστημα. Συνήθως τα logs αυτά αποθηκεύονται σε αρχεία στο τοπικό σύστημα αρχείων. Καθώς το σύστημα μεγαλώνει και αποκτά περισσότερους κόμβους, η παρακολούθηση των logs και η εξαγωγή της απαραίτητης πληροφορίας από αυτά είναι μία αρκετά δύσκολη διαδικασία. Εξάλλου όπως προαναφέρθηκε, τα logs είναι μία πηγή δεδομένων στον κλάδο της ΑΔΜΟ [1].

Προς αυτή την κατεύθυνση έχουν αναπτυχθεί διάφορα εργαλεία και συστήματα, τα οποία προσπαθούν να οργανώσουν την πληροφορία, και να δώσουν στον χρήστη το ζητούμενο, σε όσο γίνεται πιο απλή και απέριττη μορφή. Στην παρούσα διπλωματική για τα logs των εργαλείων που θα αναλυθούν στο επόμενο τμήμα, παρέχεται η ζητούμενη δυνατότητα από τα ίδια τα εργαλεία. Όσο αφορά το σύστημα και τις διάφορες μετρικές του, χρησιμοποιείται το Ganglia 2.3.1. Επιπρόσθετα, για την γρήγορη και άμεση παροχή κάποιων μετρικών, υλοποιούνται scripts 2.3.2 τα οποία έχουν συγκεκριμένο σκοπό.

2.3.1 Ganglia

Το Ganglia [14] είναι ένα κλιμακώσιμο (scalable) σύστημα με σκοπό την παρακολούθηση υψηλών επιδόσεων συστημάτων, όπως για παράδειγμα το σύμπλεγμα μηχανών που χρησιμοποιείται στην παρούσα διπλωματική. Αναπτύχθηκε στο UCB [2]. Είναι βασισμένο σε ένα πρωτόκολλο τύπου multicast listen/announce προκειμένου να παρακολουθεί την κατάσταση του συμπλέγματος, καθώς επίσης και χρησιμοποιεί έναν δέντρο από σημείο-προς-σημείο συνδέσεις, ανάμεσα στους διάφορους κόμβους του συμπλέγματος, ώστε να συναθροίσει την

[1] Ανάλυσης Δεδομένων Μεγάλου Όγκου
[2] University of California, Berkeley

κατάστασή τους. Χρησιμοποιεί τεχνολογίες όπως XML για απεικόνιση δεδομένων, XDR για μεταφορά και RRD για την αποθήκευση και εικονοποίηση τους. Χρησιμοποιεί προσεκτικά επιλεγμένες δομές δεδομένων και αλγορίθμους, προκειμένου να πετύχει χαμηλή επίπτωση ανά κόμβο και υψηλό ταυτοχρονισμό. Η υλοποίηση έχει μεταφερθεί σε ένα μεγάλο πλήθος αρχιτεκτονικών και λειτουργικών συστημάτων με επιτυχία, σε πάνω από 500 συμπλέγματα ανά τον κόσμο.

Σχετικά με την αρχιτεκτονική του ganglia:

gmond
: Το Ganglia MONitor Daemon είναι μία διεργασία που συλλέγει δεδομένα και πρέπει να εγκατασταθεί σε κάθε κόμβο του συμπλέγματος. Το gmond παίρνει μετρήσεις από τους κόμβους, και στέλνει την πληροφορία στους άλλους κόμβους με XML. Οι μετρήσεις αυτές αφορούν χαρακτηριστικά του συστήματος όπως χρησιμοποίηση του επεξεργαστή (CPU utilization), της μνήμης (Memory utilization), των δίσκων (disk utilization), και του δικτύου (I/O requests and Bytes in/out).

gmetad
: Το Ganglia METAdata Daemon είναι μία διεργασία που παρέχει έναν μηχανισμό αναζήτησης για την συλλογή ιστορικών πληροφοριών σχετικά με ένα σύνολο από μηχανήματα. Η διεργασία αυτή, συνήθως εγκαθίσταται σε έναν εξυπηρετητή (server) αν και πολύ μεγάλα συμπλέγματα είναι πιθανό να έχουν παραπάνω από μία τέτοια διεργασία. Το gmetad παίρνει τα δεδομένα του από άλλες διεργασίες gmetad και gmond και αποθηκεύει την κατάστασή του σε indexed RRD βάσεις, όπου μία web - διαπροσωπεία διαβάζει και επιστρέφει πληροφορίες σχετικά με το σύμπλεγμα.

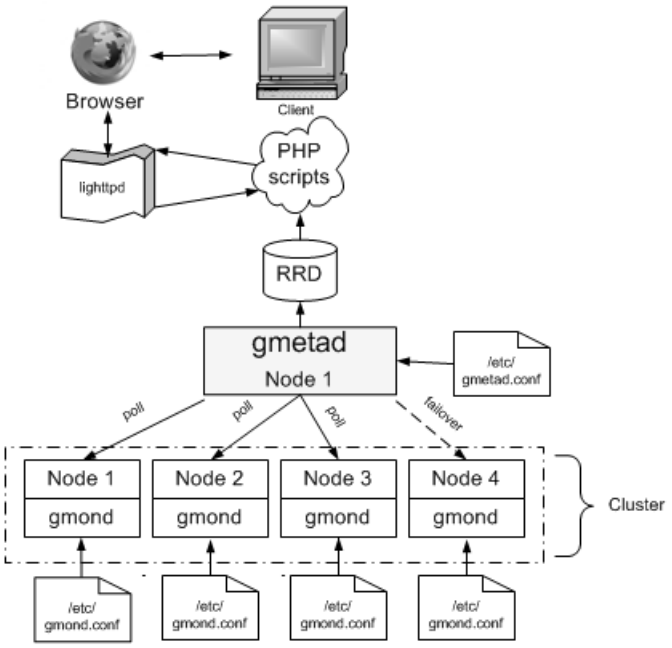
RRDtool
: Το εργαλείο αυτό είναι υπεύθυνο για το logging και για το γραφικό σύστημα που χρησιμοποιεί το Ganglia προκειμένου να αποθηκεύσει τα συγκεντρωμένα δεδομένα του και να δημιουργήσει τα γραφήματα για διαδικτυακού τύπου αναφορές.

PHP-based Web interface
: Μία διαπροσωπεία που περιέχει ένα σύνολο από scripts, σκοπός των οποίων είναι να παίρνουν τα δεδομένα από την RRD βάση και να δημιουργούν XML γραφήματα.

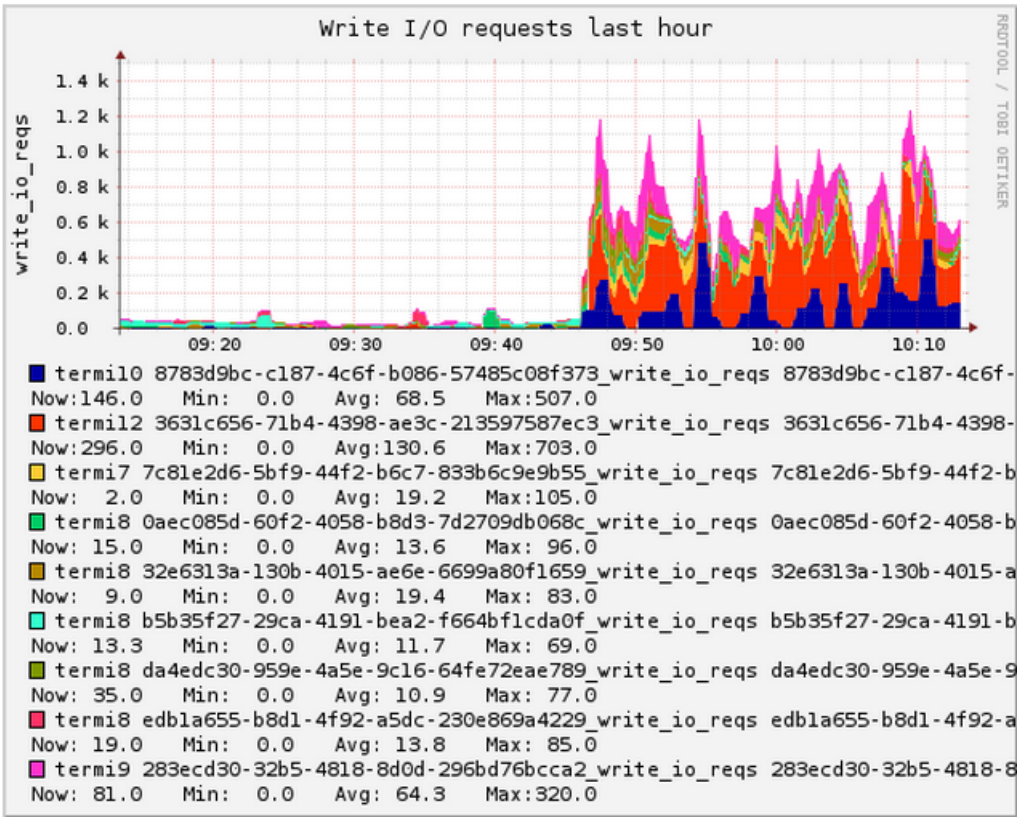
Web Server
: Ο εξυπηρετητής ιστού, χρησιμοποιεί το lighttpd, έναν http server ο οποίος μπορεί να είναι οποιοσδήποτε εξυπηρετητής που υποστηρίζει PHP, SSL, και XML. Το ganglia χρησιμοποιεί PHP για την απεικόνιση των δεδομένων με γραφικό τρόπο.

Advanced Tools
: Το gmetric είναι ένα εκτελέσιμο το οποίο προσδίδει επιπρόσθετα στατιστικά και χρησιμοποιείται για να αποθηκεύσει μετρικές που ορίζει ο χρήστης.

Το ganglia είναι εγκατεστημένο στο νέφος του OpenStack, και δίνει έναν πολύ εύχρηστο τρόπο παρακολούθησης διαφόρων μετρικών. Το χρονικό διάστημα παρακολούθησης είναι προσδιοριζόμενο με ακρίβεια δευτερολέπτου από τον χρήστη.



Σχήμα 2.5: Αρχιτεκτονική του Ganglia



Σχήμα 2.6: Παρακολούθηση με Ganglia

2.3.2 Scripts

Το παρακάτω script στοχεύει στην εκτύπωση της χρησιμοποιούμενης μνήμης RAM κατά μέσο όρο σε megabyte, εάν αφαιρεθεί ο χώρος από τη χρησιμοποίηση των buffer και της cache. Το script ολοκληρώνει την λειτουργία του όταν λάβει σήμα SIGTERM.

```
#!/bin/bash

on_die()
{
        res=$((accum/count))
        echo "Average Memory Used:"$res
        temp=$(free -m | awk '/-/ {print $3;}')
        diff=$((res-temp))
        echo "Difference from Previous Usage:"$diff

        # Need to exit the script explicitly when done.
        # Otherwise the script would live on, until system
        # realy goes down, and KILL signals are send.
        #
        exit 0
}


# Execute function on_die() receiving TERM signal
#
trap 'on_die' INT SIGTERM

# Loop forever, monitoring memory
#
accum=0
count=0
while true ; do
        temp=$(free -m | awk '/-/ {print $3;}') #measure -/+ buffers/cache: used
        accum=$((accum+temp))
        count=$((count+1))
        sleep 1
done

# We never get here.
exit 0
```

Listing 2.1: Script για την χρησιμοποίηση μνήμης

2.4 Μηχανές και Εργαλεία Εκτέλεσης

Η ανάπτυξη που έχει γνωρίσει ο κλάδος ΑΔΜΟ, μπορεί εύκολα να επιβεβαιωθεί από την πληθώρα εργαλείων και μηχανών και είναι διαθέσιμα στον χρήστη αλλά και εξελίσσονται συνεχώς, βελτιώνονται τις επιδόσεις τους και προσαρμόζοντας τη λειτουργία τους ανάλογα με την κατάσταση της τεχνολογίας. Η κάθε μία από αυτές τις μηχανές είναι στοχευμένη για κάποιο/α είδη εφαρμογών. Ταυτόχρονα πολλές από αυτές διαθέτουν εύχρηστους τρόπους επικοινωνίας και συμβατότητας με άλλες μηχανές και εργαλεία.

Οι μηχανές που επιλέχτηκαν για τους σκοπούς της παρούσας διπλωματικής είναι η σχεσιακή βάση δεδομένων postgreSQL 2.4.1, το Hadoop 2.4.2, το Hive 2.4.3, και τέλος το Spark 2.4.4. Ακολουθεί μία σύντομη ανάλυση της κάθε μίας, καθώς και μία νύξη για τον λόγο που επιλέχτηκε.

2.4.1 PostgreSQL

Η PostgreSQL [17] είναι μία αντικειμενοστραφής σχεσιακή βάση δεδομένων που αναπτύχθηκε από το UCB. Η postgreSQL είναι ανοιχτού κώδικα λογισμικό, και υποστηρίζει ένα μεγάλο μέρος απο SQL standard καθώς επίσης και πολλά μοντέρνα χαρακτηριστικά:

- Πολύπλοκα ερωτήματα (queries)
- Ξένα κλειδιά (foreign keys)
- Triggers
- Ανανεώσιμες όψεις (updatable views)
- Ακεραιότητα συναλλαγών
- Πολλαπλών εκδόσεων έλεγχο συγχρονισμού

Όπως προκύπτει από το [15], όπως και από άλλες έγκυρα τεκμηριωμένες πηγές στο διαδίκτυο, η postgreSQL είναι μία σχεσιακή βάση η οποία υπερτερεί έναντι των άλλων δημοφιλών επιλογών (π.χ. MySQl), σε θέματα επίδοσης όσο αφορά τις συνενώσεις (joins) πινάκων. Δεδομένου ότι η παρούσα διπλωματική πραγματεύεται αλγορίθμους συνένωσης και επίδοση σε ΑΔΜΟ, θεωρήθηκε σκόπιμο να χρησιμοποιηθεί η postgreSQL ως η σχεσιακή βάση δεδομένων.

2.4.2 Hadoop

Το Hadoop είναι ένα σύνολο εφαρμογών που τρέχουν σε συμπλέγματα υπολογιστών (cluster). Το Hadoop προσφέρει ένα επίπεδο αφαίρεσης στη διαδικασία ανάπτυξης παράλληλων προγραμμάτων, αναλαμβάνοντας να διαχειριστεί το διαμοιρασμό των δεδομένων, την συγκέντρωση των αποτελεσμάτων, τις πιθανές αποτυχίες κόμβων και άλλα θέματα που κανονικά θα χρειαζόταν ο προγραμματιστής να υλοποιήσει. Έργο του τελευταίου, είναι η υλοποίηση σε Java προγραμμάτων, τα οποία είναι βασισμένα στις βιβλιοθήκες του Hadoop και στην λογική των συναρτήσεων Map - Reduce [7], έτσι ώστε να εκμεταλλευτεί την παράλληλη επεξεργασία που προσφέρει το σύμπλεγμα.

Το Hadoop αποτελείται αρχιτεκτονικά από τα εξής επιμέρους στοιχεία:

Hadoop Common
: Η υπηρεσία που δίνει πρόσβαση στα συστήματα αρχείων (filesystems) που υποστηρίζει το Hadoop

Hadoop HDFS
: Ένα κατανεμημένο σύστημα αρχείων, που προσδίδει υψηλό throughput στην πρόσβαση των δεδομένων της εφαρμογής

Hadoop Yarn
Ένα framework για δρομολόγηση εργασιών και διαχείριση πόρων στο σύμπλεγμα.

Hadoop MapReduce
Ένα σύστημα βασισμένο στο Yarn, που αναλαμβάνει την παράλληλη επεξεργασία για μεγάλα σύνολα δεδομένων

Η βασική ορολογία, όσο αφορά τα κύρια μέρη του Hadoop, συνοψίζεται στα εξής [2]:

NameNode
Το κεντρικό κομμάτι του HDFS. Κρατάει τον κατάλογο από τα αρχεία του συστήματος αρχείων σε δεντρική μορφή και παρακολουθεί, την τοποθεσία στο cluster που κάθε αρχείο είναι αποθηκευμένο. Δεν αποθηκεύει τα δεδομένα το ίδιο. Τρέχει στον κόμβο Master

DataNode
Αποθηκεύει την πληροφορία στο HDFS. Τρέχει σε κάθε κόμβο slave και είναι συνδεδεμένος με τον NameNode.

Resource Manager
Αποτελείται από δύο βασικά μέρη:

- Scheduler : Είναι υπεύθυνος για την συγκέντρωση των απαραίτητων πόρων με σκοπό την εκτέλεση εφαρμογών.
- ApplicationsManager : Είναι υπεύθυνος για την αποδοχή των job, αφού διαπραγματευτεί το πρώτο container για την εκτέλεση του συγκεκριμένου για την εφαρμογή ApplicationMaster. Διαθέτει επίσης υπηρεσία επανεκκίνησης του ApplicationMaster σε περίπτωση βλάβης.

Ο Resource Manager τρέχει στον κόμβο Master.

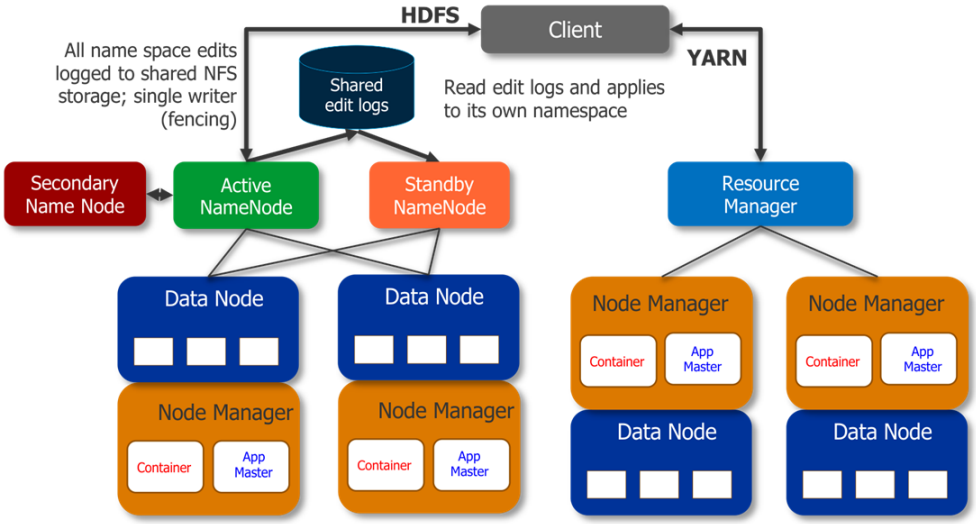
NodeManager
Είναι ένα framework που είναι υπεύθυνος για τα containers, παρακολουθώντας την χρησιμοποίηση πόρων και αναφέροντας τα πορίσματα στον ResourceManager. Τρέχει στους κόμβους slave.

Container
Είναι μια συλλογή από φυσικούς πόρους, που έχει δωθεί σε κάποιο / κάποιους κόμβους.

ApplicationMaster
Είναι υπεύθυνος για την διαπραγμάτευση των απαραίτητων container με τον Scheduler. Παρακολουθεί την κατάσταση και πρόοδο τους.

Ένας από τους κυριότερους λόγους της επιτυχίας αυτού του framework είναι το γεγονός ότι σχεδιάστηκε να τρέχει σε συστάδες από απλούς υπολογιστές (clusters) χαμηλού κόστους όπου όμως η συχνότητα βλαβών είναι υψηλή. Οι βλάβες αντιμετωπίζονται αυτόματα από το framework εξασφαλίζοντας τη σταθερότητα του συστήματος, ενώ παράλληλα η διαθεσιμότητα (availability) των δεδομένων από το HDFS δεν επηρεάζεται.

Το HDFS επιτυγχάνει να είναι αξιόπιστο, αποθηκεύοντας τα δεδομένα και αντίγραφα αυτών σε περισσότερους από ένα κόμβους. Συγκεκριμένα, ο χώρος των αρχείων είναι ενιαίος για όλο το cluster με τα αρχεία να διασπώνται σε blocks με κάθε block να αντιγράφεται σε



Σχήμα 2.7: Αρχιτεκτονική του Hadoop 2 και YARN

πολλαπλούς κόμβους δεδομένων. Μεταξύ των κόμβων υπάρχει επικοινωνία για εξισορρόπηση των δεδομένων, αντιγραφή ή μετακίνηση τους, έτσι ώστε να παραμείνει ψηλός ο λόγος αντιγραφής (replication).

Μοναδικό σημείο αποτυχίας του HDFS (single point of failure) αποτελεί ο κεντρικός κόμβος, ο name node, ο οποίος διατηρεί πληροφορίες σχετικά με το που βρίσκονται τα δεδομένα στο HDFS και στη περίπτωση που δεν είναι διαθέσιμος δεν είναι δυνατή η πρόσβαση σε αυτά. Ένας δεύτερος κόμβος, ο secondary namenode, διατηρεί αντίγραφα των φακέλων του name node (snapshots) και μαζί με τα αρχεία χρήσης του name node (logs) είναι δυνατή η επαναφορά του συστήματος σε περίπτωση κάποιας βλάβης. Οι κόμβοι όπου αποθηκεύονται δεδομένα ονομάζονται datanodes.

Το Hadoop είναι ίσως το πιο ευρέως γνωστό και χρησιμοποιούμενο εργαλείο στην ΑΔΜΟ. Η επίδοσή του είναι πολύ ανταγωνιστική [2], ενώ ταυτόχρονα, πολλά άλλα εργαλεία χρησιμοποιούν πλέον το Hadoop framework [9], όπως το Apache Pig, Apache Hive, Apache HBase, Apache Spark δημιουργώντας μια ισχυρή πλατφόρμα με πολλαπλές επιλογές για την ανάλυση και επεξεργασία δεδομένων σε κατανεμημένα περιβάλλοντα.

2.4.3 Hive

Το Apache Hive [6] είναι μία αποθήκη δεδομένων που διευκολύνει την αναζήτηση και τη διαχείριση μεγάλων συνόλων δεδομένων, ενός συστήματος κατανεμημένης αποθήκευσης. Παρέχει ένα μηχανισμό για να προσδίδει δομή σε δεδομένα και ύστερα τη δυνατότητα ερωτημάτων σε αυτά, μέσω μίας γλώσσας ερωτημάτων τύπου SQL, την HiveQL. Είναι αρκετά επεκτάσιμη, με τον χρήστη να μπορεί να ορίσει τις δικές του συναρτήσεις column transformation (UDF) σε Java. Ταυτόχρονα, η γλώσσα αυτή προσφέρει τη δυνατότητα προσθήκης mapper και reducer[3] όταν είναι ανέφικτο ή μη αποδοτικό να εκφραστεί η αντίστοιχη λογική στην HiveQL.

Το HIVE αποτελείται αρχιτεκτονικά από τα εξής επιμέρους στοιχεία:

[3] όπως ορίζονται στο framework του MapReduce

UI

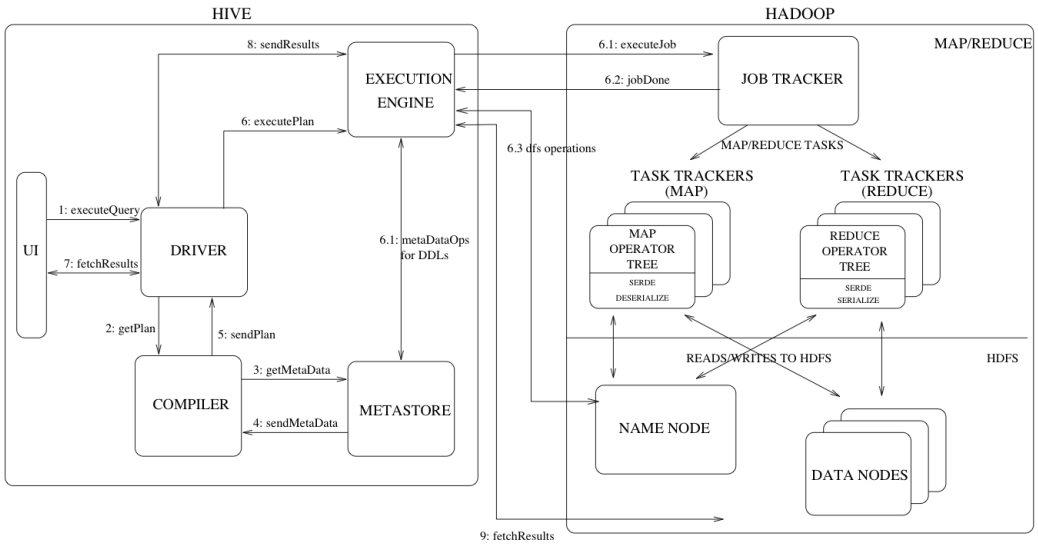
- Hive CLI: Δυνατότητα εκτέλεσης εντολών μέσω ενός CLI [4] σε λειτουργία διαδραστική ή batch.
- Web GUI: Εναλλακτικός τρόπος για εκτέλεση εντολών από αυτών του CLI. Είναι μία web - based διαπροσωπεία που δίνει ένα απλό και φιλικό προς τον χρήστη γραφικό τρόπο εκτέλεσης εντολών αλλά και απεικόνισης των υπαρχόντων πινάκων δεδομένων.

Driver
Το στοιχείο που δέχεται τα ερωτήματα. Υλοποιεί την διαπροσωπεία εκτέλεσης και

Compiler
Κάνει parse το ερώτημα, το αναλύει σημασιολογικά σε διάφορα block και expressions και τελικά καταλήγει σε ένα πλάνο εκτέλεσης με τη βοήθεια του πίνακα και του partition metadata που παίρνει από το metastore.

Metastore
Αποθηκεύει τα metadata για τους πίνακες του Hive σε μια σχεσιακή βάση, και δίνει στου χρήστες την δυνατότητα πρόσβασης στην πληροφορία μέσω ενός API. Η σχεσιακή βάση που χρησιμοποιείται είναι η derbyDB. Αποθηκεύει πληροφορίες για τους πίνακες και τα partitions σχετικά με τη δομή, τις στήλες και τον τύπο τους, τους σειριοποιητές και αποσειριοποιητές που είναι απαραίτητη για την ανάγνωση και εγγραφή των δεδομένων.

Execution Engine
Το στοιχείο αυτό, είναι υπεύθυνο για την εκτέλεση του πλάνου που έβγαλε ο compiler. Αυτό είναι ένα κατευθυνόμενο ακυκλικό γράφημα (DAG[5]) από στάδια. Το execution engine διαχειρίζεται τα dependecies ανάμεσα στα στάδια, και τα εκτελεί με τα κατάλληλα στοιχεία του συστήματος.



Σχήμα 2.8: Αρχιτεκτονική του HIVE

Τα δεδομένα του Hive είναι οργανωμένα σε:

[4] Command Line Interface
[5] Directed Acyclic Graph

1. **Tables**: Οι πίνακες αυτοί είναι ανάλογοι με τους πίνακες που γνωρίζουμε από τις σχεσιακές βάσεις δεδομένων. Κάθε πίνακας έχει ένα αντίστοιχο directory στο HDFS όπου είναι αποθηκευμένα τα δεδομένα. Τα δεδομένα σειριοποιούνται και αποθηκεύονται σε διάφορα αρχεία μέσα σε αυτό το directory.Το Hive προσφέρει διάφορες μορφές σειριοποίησης που διευκολύνουν τη συμπίεση των δεδομένων, ενώ ο προγραμματιστής έχει τη δυνατότητα να ορίσει ακόμα και τις δικές του μεθόδους σειριοποίησης.

2. **Partitions**: Κάθε πίνακας μπορεί να έχει ένα ή περισσότερα partitions που προσδιορίζουν την περαιτέρω κατανομή των δεδομένων σε υπο-φακέλους. Το πλεονέκτημα είναι ότι όλα τα δεδομένα που έχουν την ίδια τιμή στη στήλη που έχει χρησιμοποιηθεί για το partition καταλήγουν στο ίδιο sub-directory και υπάρχει η δυνατότητα να εκτελέσουμε κάποιο ερώτημα σε αυτά τα δεδομένα χωρίς να χρειαστεί να επεξεργαστούμε το σύνολο των δεδομένων.

3. **Buckets**: Σε κάθε partition τα δεδομένα μπορούν να διαχωριστούν περαιτέρω σε buckets χρησιμοποιώντας την τιμή κατακερματισμού μιας στήλης του πίνακα. Κάθε bucket αποθηκεύεται ως ένα αρχείο στο directory του partition.

Το HIVE επιλέχθηκε, προκειμένου να δώσει μία δομή (schema) στα δεδομένα που είναι αποθηκευμένα στο HDFS. Όπως προαναφέρθηκε, το hive έχει εγγενώς έναν πολύ εύχρηστο τρόπο αναφοράς σε δεδομένα που είναι αποθηκευμένα στο HDFS. Έτσι τα δεδομένα αυτά είναι πλέον οργανωμένα σε πίνακες, χωρίς ουσιαστικό επιπλέον κόστος, και μπορούν να εκτελεστούν SQL ερωτήματα πάνω σε αυτά, και πιο συγκεκριμένα, να εφαρμοστούν αλγόριθμοι συνένωσης.

2.4.4 Spark

Το Apache Spark είναι ένα σύστημα γενικού σκοπού, για υπολογισμούς σε συμπλέγματα μηχανημάτων. Είναι γνωστό για την ταχύτητα των υπολογισμών του, καθώς και για την κλιμακοσιμότητα τους. Το spark είναι ακόμα ένα εργαλείο, που προσφέρεται ως εναλλακτική του MapReduce αλλά ωστόσο χρησιμοποιώντας το HDFS ως σύστημα αποθήκευσης μεγάλων αρχείων. Τα κύρια πλεονεκτήματα που κάνουν το spark να υπερτερεί έναντι άλλων εργαλείων σε πολλές εφαρμογές είναι [16]:

- Παρέχει ένα ενοποιημένο framework για να διαχειρίζεται δεδομένα μεγάλου όγκου, με εντελώς διαφορετικό περιεχόμενο (κειμένου, γραφικά κτλ) όπως επίσης και το είδος της πηγής (πραγματικού χρόνου - batch).

- Επιτρέπει σε εφαρμογές σε συμπλέγματα Hadoop να τρέχουν μέχρι και 100 φορές γρηγορότερα στην μνήμη και 10 φορές γρηγορότερα ακόμα και όταν χρησιμοποιούν το δίσκο.

- Επιτρέπει την εύκολη και γρήγορη δημιουργία εφαρμογών σε Java, Scala και Python.

- Διαθέτει βιβλιοθήκες με πολύ χρήσιμα χαρακτηριστικά:

  - **Spark Streaming**: Δυνατότητα για επεξεργασία δεδομένων σε πραγματικό χρόνο. Είναι βασισμένη σε micro batch style επεξεργασίας και υπολογισμού.

- **Spark SQL**: Η βιβλιοθήκη αυτή επιτρέπει, με χρήση του κατάλληλου JDBC, την εξαγωγή - διαμόρφωση - φόρτωση δεδομένων διαφόρων τύπων (JSON, Parquet, Database), την μεταμόρφωσή τους, και τέλος την διεξαγωγή ερωτημάτων τύπου SQL. Η βιβλιοθήκη αυτή δίνει πλήρη συμβατότητα του Spark με το Hive, και κατά συνέπεια πρόσβαση στα δεδομένα του.
- **Spark MLlib**: Κλιμακόσιμη βιβλιοθήκη για machine learning, που περιέχει του συνήθεις αλγορίθμους και εργαλεία, συμπεριλαμβανομένου classification, regression, clustering, collaborative filtering, dimensionality reduction, καθώς επίσης και underlying optimization primitive.
- **Spark GraphX**: Το νέο API του Spark για υπολογισμούς γραφημάτων. Περιέχει μια συνεχώς αυξανόμενη συλλογή από αλγορίθμους γραφημάτων και μεθόδους για την απλοποίηση των διεργασιών.

Για την υποστήριξη διάφορων λειτουργιών του, το Spark χρησιμοποιεί μία δομή που ονομάζεται Resilient Distributed Dataset (RDD) και αποτελεί ουσιαστικά ένα σύνολο αντικειμένων μόνο για ανάγνωση, μοιρασμένη στο cluster υπολογιστών. Χρησιμοποιεί τα δεδομένα του HDFS. Επίσης βασικό χαρακτηριστικό της είναι ότι όταν ο προγραμματιστής - χρήστης, εφαρμόσει διάφορους μετασχηματισμούς σε ένα RDD, στη μνήμη τελικά αποθηκεύεται μόνο η ακολουθία των μετασχηματισμών αυτών, και όχι τα δεδομένα που προκύπτουν. Μόνο στην περίπτωση κάποιας ειδικής εντολής - ενέργειας (π.χ. reduce, collect, count κτλ) γίνεται η αποτίμηση. Δεδομένου ότι τα σετ δεδομένων βρίσκονται (κατά το δυνατόν) στην μνήμη, σε περίπτωση σφάλματος, είναι εύκολη η ανάκτησή τους αφού υπάρχει η σειρά των μετασχηματισμών στο RDD.

Αξίζει να σημειωθεί επίσης πως το spark υιοθετεί την τεχνική οκνηρής αποτίμησης (lazy evaluation) σε ερωτήματα μεγάλου όγκου δεδομένων, καθώς επίσης και εφαρμόζει βελτιστοποιήσεις για διάφορες λειτουργίες.

Τα βασικά χαρακτηριστικά του Spark είναι τα εξής:

Data Storage

Ως μέσο αποθήκευσης, χρησιμοποιείται το HDFS.

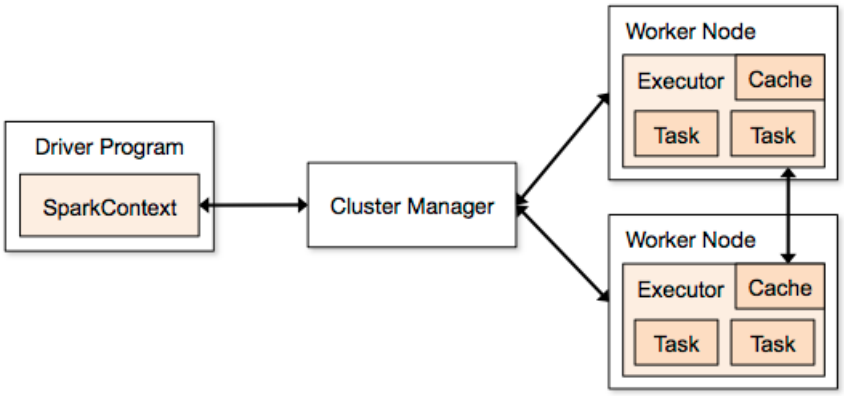
API

- Scala
- Java
- Python

Management Framework

- Stand-alone
- Mesos
- YARN

Η βασική ορολογία, όσο αφορά τα κύρια μέρη του spark, συνοψίζεται στα εξής:



Σχήμα 2.9: Αρχιτεκτονική του Spark

Application
Το πρόγραμμα του χρήστη. Αποτελείται από το driver program και τους executors.

Driver program
Η διεργασία που τρέχει στον κόμβο Master και έχει την main() μέθοδο της εφαρμογής και δημιουργεί το SparkContext.

Cluster Manager
Μία εξωτερική υπηρεσία για την συγκέντρωση των απαραίτητων πόρων για το cluster. (standalone, Mesos, YARN)

Worker Node
Οποιοσδήποτε κόμβος μπορεί να τρέξει κώδικα της εφαρμογής στο cluster.

Executor
Μια διεργασία που εκκινείται σε έναν worker node, εκτελείται και κρατά τα δεδομένα της στην μνήμη ή/και στον δίσκο. Κάθε application έχει τους δικούς της executors.

Task
Ένα κομμάτι εργασίας, που θα αποσταλεί σε έναν executor.

Job
Ένας παράλληλος υπολογισμός, που αποτελείται από πολλαπλά tasks.

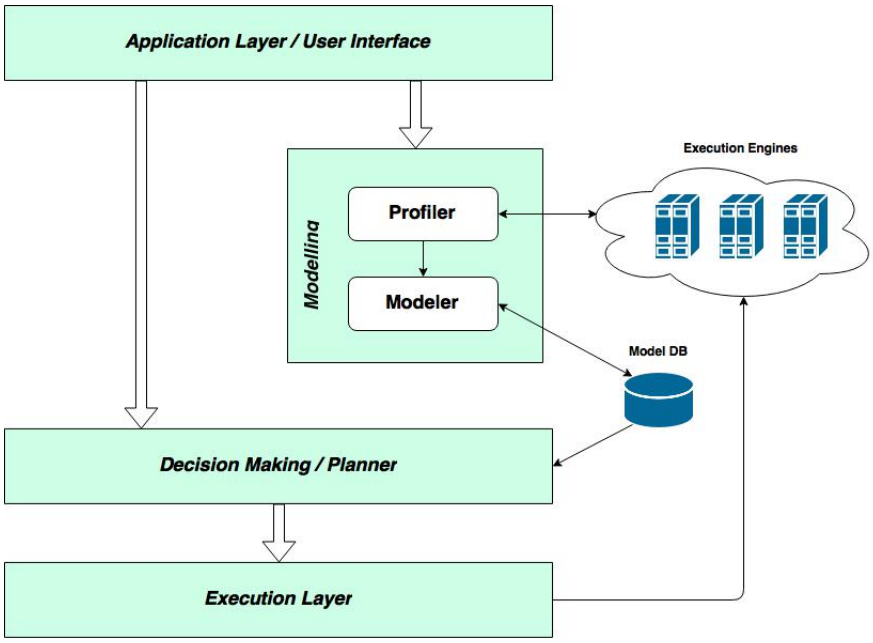
Stage
Κάθε Job διαιρείται σε μικρότερα σύνολα από tasks που λέγονται stages και εξαρτιούνται το ένα από το άλλο.

Κεφάλαιο 3

Σχεδιασμός Συστήματος

Σε αυτό το κεφάλαιο δίνονται λεπτομέρειες σχετικά με τον σχεδιασμό του συστήματος που πραγματεύεται αυτή η διπλωματική. Πιο συγκεκριμένα, στο Τμήμα 3.1 αναλύεται η βασική αρχιτεκτονική του συστήματος. Ακολουθεί στο Τμήμα 3.2 περιγράφεται η διαδικασία που πραγματοποιείται για την παραγωγή των ζητούμενων μοντέλων. Στο Τμήμα 3.3 παρουσιάζεται το σύμπλεγμα των μηχανημάτων (cluster) και ο τρόπος διαχείρισής του, όπως επίσης και οι βασικές τοπολογίες για κάθε μηχανή. Στο Τμήμα 3.4 αναλύεται ο τρόπος και οι σχεδιαστικές αποφάσεις που πάρθηκαν για την δημιουργία του συνόλου δεδομένων.

3.1 Αρχιτεκτονική Συστήματος



Σχήμα 3.1: Αρχιτεκτονική Συστήματος

Το σύστημα είναι μέρος του ευφυούς δρομολογητή IReS [13] που ανήκει στην πλατφόρμα ASAP. Η βασική αρχιτεκτονική φαίνεται στο Σχ. 3.1 και αποτελείται από τα εξής επιμέρους στοιχεία:

Επίπεδο Εφαρμογής - Χρήστη (Application Layer / User Interface)
Αποτελεί την διαπροσωπεία με τον χρήστη (UI), από όπου λαμβάνονται στοιχεία σχε-

τικά με την ζητούμενη εργασία, τα χαρακτηριστικά εισόδου της, καθώς και ο στόχος της όσο αφορά την βελτιστοποίηση της επίδοσής της.

Μοντελοποίηση (Modelling)
Είναι το επίπεδο στο οποίο συνεισφέρει η παρούσα διπλωματική. Σκοπός του είναι η παραγωγή μοντέλων επίδοσης για τις υπάρχουσες μηχανές εκτέλεσης, και αποθήκευσή τους σε μία βάση μοντέλων (Model DB). Περιέχει δύο επιμέρους δομικά στοιχεία:

1. **Profiler**
   Σκοπός αυτού του μέρους είναι η εκτέλεση εργασιών με κάθε συνδυασμό παραμέτρων εισόδου, για κάθε πιθανή μετρική βελτιστοποίησης του χρήστη. Οι παράμετροι εισόδου του χρήστη χωρίζονται σε 3 κατηγορίες:
   - Παράμετροι συνόλων δεδομένων (*Data Specific Parameters*): Περιγράφουν τα δεδομένα που χρησιμοποιούνται από τον profiler
   - Παράμετροι πόρων (*Resource Specific Parameters*): Καθορίζουν τους πόρους που ρυθμίζονται κατά τις εκτελέσεις των εργασιών.
   - Παράμετροι αλγορίθμων (*Algorithm Specific Parameters*): Σχετίζονται με τους αλγορίθμους που εξετάζονται.
2. **Modeler**
   Το κομμάτι αυτό, είναι υπεύθυνο για την δημιουργία μοντέλων για κάθε παράμετρο εισόδου, για κάθε μετρική βελτιστοποίησης του χρήστη και για κάθε διαθέσιμη μηχανή εκτέλεσης. Επίσης αναλαμβάνει την αποθήκευσή τους στη βάση μοντέλων (model DB).

Επίπεδο Απόφασης και Σχεδίασης Εκτέλεσης (Decision Making / Planner)
Σε αυτό το επίπεδο, εξετάζεται η είσοδος του χρήστη, σε συνδυασμό με τα υπάρχοντα στη βάση μοντέλα και σχεδιάζεται το βέλτιστο πλάνο εκτέλεσης της εργασίας του χρήστη.

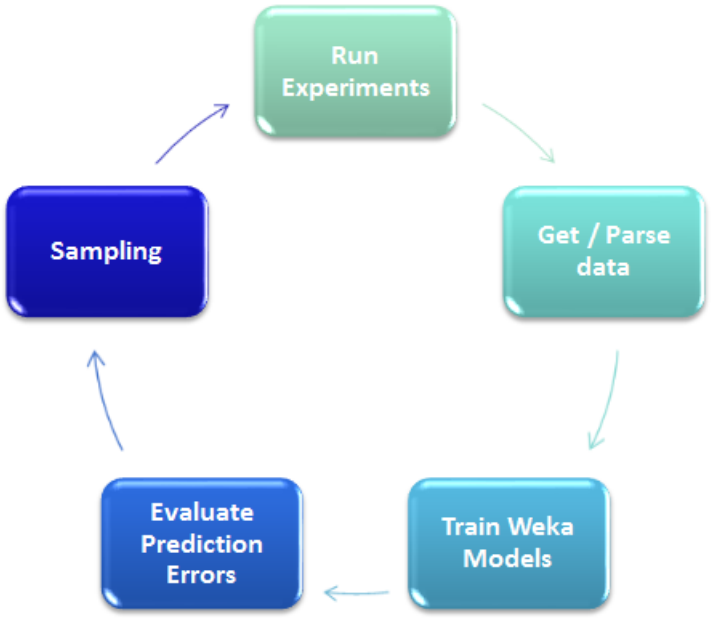
Επίπεδο Εκτέλεσης (Execution Layer)
Εκτελείται το πλάνο στις μηχανές εκτέλεσης της υποδομής συμπλέγματος (cluster).

3.2 Διαδικασία Μοντελοποίησης

Η διαδικασία, με την οποία παράγονται τα ζητούμενα από το σύστημα μοντέλα και πρακτικά υλοποιούνται οι λειτουργίες του profiler και modeler, είναι επαναληπτική και φαίνεται από το Σχ. 3.2

Πιο αναλυτικά:

1. **Εκτέλεση πειραμάτων** (Run Experiments) :
   Στην φάση αυτή, επιλέγεται το βέλτιστο configuration για τις μηχανές, ανάλογα με τους διαθέσιμους πόρους του χρήστη. Η επιλογή γίνεται με βάση κάποιους κανόνες για την επιλογή των σημαντικότερων για την επίδοση παραμέτρων [1]. Στην συνέχεια πραγματοποιείται η εκτέλεση των εργασιών στο cluster.
2. **Ανάκτηση Μετρήσεων** (Get / Parse Data) :
   Στην φάση αυτή, έχει ολοκληρωθεί η εκτέλεση των εργασιών, και γίνεται η ανάκτηση των δεδομένων από τους μηχανισμούς παρακολούθησης που χρησιμοποιούνται ανά περίπτωση.



Σχήμα 3.2: Διαδικασία Μοντελοποίησης

3. **Εκπαίδευση των μοντέλων του Weka** (Train Weka Models) :
   Στην φάση αυτή, χρησιμοποιείται το εργαλείο Weka με τα υπάρχοντα δεδομένα, ώστε να επιλεχτεί, να εκπαιδευτεί και να γίνει όσο το δυνατόν πιο έγκυρο (μικρότερο σφάλμα) το μοντέλο.

4. **Εκτίμηση σφαλμάτων πρόβλεψης** (Evaluate Prediction Errors) :
   Στην φάση αυτή, χρησιμοποιείται το εργαλείο Weka ώστε να εκτιμηθούν τα σφάλματα πρόβλεψης του χρησιμοποιούμενου μοντέλου πρόβλεψης.

5. **Δειγματοληψία** (Sampling) :
   Χρησιμοποιούνται εποπτικές τεχνικές για την αποτελεσματική κάλυψη του δειγματοχώρου, οι οποίες βασίζονται στα εκάστοτε τρέχοντα αποτελέσματα.

Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου θεωρηθεί ότι τα μοντέλα είναι αρκετά έγκυρα στα πλαίσια αυτής της διπλωματικής. Σε αυτό το σημείο αξίζει να σημειωθεί, ότι θεωρήθηκε σκόπιμο να αναλυθούν σε μεγαλύτερο βάθος κάποια μοντέλα και να προκύψουν κατα το δυνατόν εγκυρότερα, παρά να καλυφθεί ο δειγματικός χώρος στο σύνολό του.

3.3 Διαχείριση συμπλέγματος (cluster)

Ο κλάδος ΑΔΜΟ, είναι ιδιαίτερα απαιτητικός όσο αφορά τους υπολογιστικούς πόρους. Για αυτό τον λόγο και δεν είναι δυνατή η διεξαγωγή μετρήσεων και πειραμάτων σε μεμονωμένα μηχανήματα. Την λύση σε αυτό το πρόβλημα, καθώς και γενικά μία ισχυρή ώθηση του κλάδου, έδωσαν οι πλατφόρμες υπολογιστικού νέφους (cloud computing platforms).

Τα μηχανήματα σε αυτό το πλαίσιο είναι εικονικά (virtual machines), και το επίπεδο του υλικού είναι σε εικονική μορφή, υλοποιημένο από λογισμικό. Ο χρήστης, μπορεί να συνδέεται στα μηχανήματά του, από οποιοδήποτε υπολογιστή ή συσκευή έχει σύνδεση στο διαδίκτυο.

Εξού και το όνομα ”νέφους”. Ταυτόχρονα οι απαιτήσεις πόρων για το χρήστη είναι ελάχιστες (απλά μια σύνδεση στο διαδίκτυο). Τα φυσικά μηχανήματα και οι αρχιτεκτονικές λεπτομέρειες δόμησης και διασύνδεσης τους δεν αφορούν τον χρήστη. Είναι χαρακτηριστικά της υποδομής που του δίνεται ως υπηρεσία (IaaS). Το γεγονός αυτό προσδίδει τεράστια ευελιξία και κλιμακοσιμότητα.

Η υποδομή που επιλέχθηκε για την εναπόθεση του του συμπλέγματος μηχανών (cluster) είναι το **OpenStack**. Το OpenStack είναι λογισμικό ανοιχτού κώδικα που δίνει τη δυνατότητα για δημιουργία και διαχείριση τόσο σε ιδιωτικό όσο και σε δημόσιο επίπεδο, πλατφορμών υπολογισμών νέφους. Το OpenStack δίνει τη δυνατότητα διαχείρισης με χρήση ενός ταμπλό (dashboard) και μέσω μίας διαπροσωπίας προγραμματισμού εφαρμογών (API), προσβάσιμων μέσω διαδικτύου.

Το σύννεφο το οποίο χρησιμοποιείται, είναι ιδιωτικό και είναι υλοποιημένο σε μηχανήματα του εργαστηρίου [1]. Οι λεπτομέρειες της αρχιτεκτονικής των μηχανημάτων αυτών είναι οι ακόλουθες:

- 4 x Supermicro servers

  CPU : 2 x Intel(R) Xeon(R) CPU E5645 @ 2.40GHz (24 Hardware threads/server)
  RAM : 96GB
  Storage : 4 x SATA slots (2.5”)
  - 1TB για rootfs και VM images
  - 2 x 1TB για volumes (RAID 0)
  - 1 x 256GB SSD

  Network : 2 x Ethernet interfaces (μία δημόσια και μία ιδιωτική)
  Hypervisor : qemu (kvm enabled) version 2.0.0

- 2 x Supermicro servers

  CPU : 2 x Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz (32 Hardware threads/server)
  RAM : 64GB
  Storage : 4 x SATA slots (2.5”)
  - 2TB για rootfs και VM images

  Network : 2 x Ethernet interfaces (μία δημόσια και μία ιδιωτική)
  Hypervisor : qemu (kvm enabled) version 2.0.0

Το σύνολο των πόρων για όλο το ιδιωτικό σύννεφο είναι:

CPU : 160 επεξεργαστικές μονάδες

RAM : 512GB

Storage :

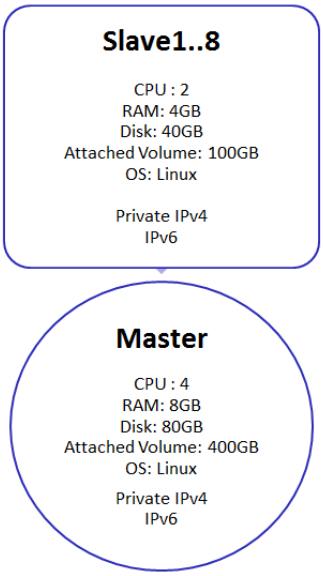
- 6TB για rootfs και VM images
- 8TB για volumes

[1] CSLab NTUA

- 1TB SSD

Παρόλο που οι πόροι σε ένα πρώτο επίπεδο φαίνονται μεγάλοι σε μέγεθος, θα πρέπει να ληφθεί σοβαρά υπόψιν ότι τα διάφορα συστήματα και εργαλεία που υπάρχουν στον χώρο της ΑΔΜΟ, είναι ιδιαίτερα απαιτητικά σε υπολογιστικούς πόρους, καθώς οι διεργασίες που επιτελούν είναι υπολογιστικά πολύ απαιτητικές. Επίσης, δεδομένου ότι τους πόρους αυτούς τους μοιράζονται αρκετοί χρήστες με παρόμοια απαιτητικά διεργασίες, είναι γεγονός ότι προκύπτουν προβλήματα έλλειψης πόρων.

Η ακόλουθη εικόνα δείχνει την πληροφορία σχετικά με το μέγεθος των πόρων που ήταν διαθέσιμα για την εν λόγω διπλωματική. Αξίζει να σημειωθεί επίσης, ότι λόγω έλλειψης διευθύνσεων IPv4 η σύνδεση στα VM [2] γίνεται με VPN [3].



Σχήμα 3.3: Παράδειγμα Τοπολογίας Cluster

Δεδομένου ότι τα εικονικά μηχανήματα του cluster είναι διαχειρίσιμα από το API και το dashboard του OpenStack, οι αρχιτεκτονικές αυτές λεπτομέρειες είναι ρυθμίσιμες, χωρίς ωστόσο το σύνολο των πόρων να μπορεί να ξεπεράσει τα όρια που εύκολα διαπιστώνει κανείς από το παράδειγμα.

Η διαχείριση των μηχανημάτων από πλευράς λογισμικού, γίνεται από τον κόμβο master. Προκειμένου να είναι εύκολη και αποδοτική αυτή η διαχείριση, υλοποιήθηκαν διάφορα bash scripts, τα βασικά χρησιμοποιούμενα των οποίων είναι:

```
#!/bin/bash

if [ "$1" = "" ] || [ "$2" = "" ] || [ "$3" = "" ] ; then
        echo "usage: $0 <Start_node> <End_node> <command>"
        exit 1
fi

pdsh -R ssh -w slave[$1-$2] "$3"
```

Listing 3.1: Script για εκτέλεση εντολών σε ένα υποσύνολο των slaves

[2] virtual machines
[3] Virtual Private Network

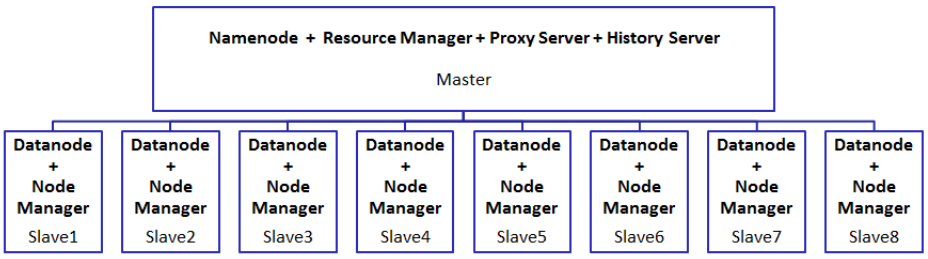
```
#!/bin/bash

if [ "$1" = "" ] || [ "$2" = "" ] ; then
        echo "usage: $0 <Start_node> <End_node>"
        exit 1
fi

for i in $(seq $1 $2); do
        echo $i
        rsync -av /etc/hosts slave$i:/etc/hosts
        echo "Testing pings for slave$i"
        pdsh.sh $1 $2 "ping -c 1 slave$i  | grep loss"
done

echo "Testing pings for master"
pdsh.sh $1 $2 "ping -c 1 master  | grep loss"
```

Listing 3.2: Script για την ενημέρωση και έλεγχο σε ένα υποσύνολο των slaves σε περίπτωση αλλαγής τοπολογίας και κατα συνέπεια διευθύνσεων ip

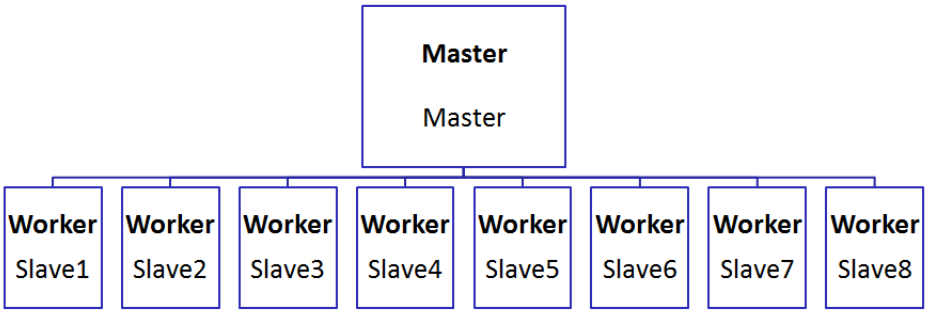
```
#!/bin/bash

if [ "$1" = "" ] || [ "$2" = "" ] ; then
        echo "usage: $0 <Start_node> <End_node>"
        exit 1
fi

for i in $(seq $1 $2); do
        echo $i
        rsync -av --exclude='logs/' --exclude='work/' --exclude='examples/' --exclude
    ='/downloads/' --exclude='/hive/' --exclude='/db-derby-10.11.1.1-bin/'  /opt/
    slave$i:/opt/ --delete-after

done
```

Listing 3.3: Script για συγχρονισμό σε ένα υποσύνολο των slaves του φακέλου /opt που περιέχει τα απαραίτητα για τις χρησιμοποιούμενες μηχανές αρχεία. Τα αρχεία που εξαιρούνται είτε πρέπει να είναι μοναδικά σε κάθε κόμβο είτε αρκεί να βρίσκονται στον κόμβο master

Σχετικά με τις μηχανές, οι ακόλουθες εικόνες αρκούν για τον τρόπο που εκτελούνται οι δαίμονες-διεργασίες των μηχανών που χρησιμοποιούνται.



Σχήμα 3.4: Τοπολογία Cluster για το Hadoop



Σχήμα 3.5: Τοπολογία Cluster για το Spark

3.4 Σύνολο Δεδομένων (Dataset)

Το dataset παίζει καθοριστικό ρόλο στα αποτελέσματα των μετρήσεων. Προκειμένου να μπορεί να είναι πλήρως ρυθμίσιμο (σχετικά με τα χαρακτηριστικά του) επιλέχτηκε να κατασκευάζεται προγραμματιστικά, αντί να λαμβάνεται από άλλες πηγές, στις οποίες πιθανότατα κάποια χαρακτηριστικά θα ήταν ασύμφορο να ανιχνευτούν αλλά και πιθανότατα να μην υπήρχαν dataset για όλες τις ζητούμενες τιμές παραμέτρων των χαρακτηριστικών του.

Δεδομένου ότι το σύστημα θα εξετάζει την επίδοση αλγορίθμων συνένωσης, θεωρήθηκε σκόπιμο τα dataset να είναι δύο πίνακες με τις εξής παραμέτρους:

- Αριθμός των πλειάδων - γραμμών : 10^2 - 10^9
- Αριθμός των columns - στηλών : 2 (BIGINT id, BIGINT value)
- Διακύμανση πυκνότητας δεδομένων με κατανομές :
  - Πιθανοτικές :
    - uniform
    - gaussian
  - Ντετερμινιστικές :
    - linear
- Ποσοστό ταιριάσματος κλειδιού συνένωσης : 10% - 100%

Αξίζει να σημειωθεί ότι το πεδίο id αποτελεί μοναδικό κλειδί για τον κάθε πίνακα (primary key), ενώ το κλειδί της συνένωσης είναι το πεδίο value.

3.4.1 Generator

Για την δημιουργία του dataset, υλοποιήθηκε ένας generator. Τα χαρακτηριστικά που προκείπτουν από τη σχεδίαση αυτού του generator είναι:

1. Τα dataset φτιάχνονται σε κομμάτια (chunks). Κάθε κομμάτι αποθηκεύεται σε ξεχωριστό αρχείο, και υπάρχει μία δεύτερη φάση συνένωσης των μεμονωμένων αρχείων σε ένα κεντρικό.
2. Όσος είναι ο αριθμός των πυρήνων του συστήματος, τόσα chunks φτιάχνονται παράλληλα.
3. Η πολυπλοκότητα κατασκευής είναι γραμμική ($\omega(n)$).
4. Η διαδικασία της δημιουργίας είναι πλήρως κλιμακόσιμη.
5. Η διαδικασία της δημιουργίας είναι προσαρμοζόμενη σε οποιαδήποτε αρχιτεκτονική συστήματος.

Η διαδικασία χωρίζεται σε δύο φάσεις:

Δημιουργία των chunks

Στην φάση αυτή δημιουργούνται παράλληλα (ανάλογα με τον αριθμό cpu cores) αρχεία δεδομένων βασισμένα σε κάποια επιλεγόμενη κατανομή. Τα αρχεία εξόδου είναι μορφής csv [4]. Ο χρήστης, δίνει ως είσοδο το επιθυμητό μέγεθος για τον μεγάλο και το μικρό πίνακα, όπως επίσης και το επιθυμητό βήμα με το οποίο θα φτιάχνονται τα chunks, τόσο για τον μικρό όσο και για τον μεγάλο πίνακα. Η δημιουργία των αρχείων γίνεται με τη χρήση δύο προγραμμάτων.

Το πρόγραμμα **generator.py**, δημιουργεί κάθε φορά που εκτελείται δύο αρχεία, μεγέθους αντίστοιχου των παραμέτρων που δέχεται ως είσοδο. Βασικό χαρακτηριστικό είναι ότι το δεύτερο και μικρότερο αρχείο, έχει πλειάδες, των οποίων τα values έχουν επιλεχθεί από το πρώτο αρχείο. Έτσι είναι σίγουρο ότι ολόκληρος ο δεύτερος πίνακας θα προκύψει στην έξοδο του join. Ακόμα ένα βασικό χαρακτηριστικό είναι ότι η κατανομή ρυθμίζεται στον κώδικα ανάλογα με την προτίμηση του χρήστη.

```
1  #!/usr/bin/python
2
3  import sys
4  import numpy
5  import random
6  import csv
7  import string
8  import threading
9  from multiprocessing import Process
10
11 if len(sys.argv)<8:
12   print "Usage: python generator.py <length> <step1> <step2> <id1_start> <
       id2_start> <outFile1> <outFile2>"
13   quit()
14
15 low,up = 0,100000
16 length = int(sys.argv[1])
17 step1 = int(sys.argv[2])
18 step2 = int(sys.argv[3])
19 if step2 > step1:
```

[4] comma-separated values

```
  print "step1 must be >= step2"
  quit()

id1_start = int(sys.argv[4])
id2_start = int(sys.argv[5])
out1 = sys.argv[6]
out2 = sys.argv[7]

if (id1_start<0) or (id2_start<0) or (step1<0) or (step2<0) or (length<0):
    print "Error: negative value given"
    quit()


mu, sigma = 10*length,2*length # gaussian mean and standard deviation (95% within
     2 sigma)
a = 1.01 #zipf parameter


def directWriteNumpy(outFile1,outFile2):
  #first dataset

  #***values****
  #values1=numpy.random.normal(mu,sigma,step1) #Gaussian
  #values1=numpy.random.zipf(a,step1) #Zipf
  #values1=numpy.random.uniform(0,10*length,step1) #Uniform
        values1= numpy.arange(id1_start+2*length,id1_start+step1+2*length,dtype=
    numpy.int64) #linear

  ids1= numpy.arange(id1_start,id1_start+step1,dtype=numpy.int64)

  output1= numpy.column_stack((ids1,values1))
  numpy.savetxt(outFile1,output1,fmt='%d',delimiter=",")

  #second dataset
  #get the first values of values1
  values2=values1[0:step2].copy()
  ids2= numpy.arange(id2_start,id2_start+step2,dtype=numpy.int64)
  output2= numpy.column_stack((ids2,values2))
  numpy.savetxt(outFile2,output2,fmt='%d',delimiter=",")
  return

directWriteNumpy(out1,out2)
```

Listing 3.4: generator.py

Το πρόγραμμα **parallel_generator.sh** είναι ένα bash script, το οποίο χρησιμοποιεί το generator.py για να δημιουργήσει τον κατάλληλο αριθμό αρχείων, με τις κατάλληλες παραμέτρους. Σημειωτέον, ότι με την εντολή sem καταφέρνει την παραλληλία στην δημιουργία.

```
#!/bin/bash

if [ "$1" = "" ] || [ "$2" = "" ] || [ "$3" = "" ] || [ "$4" = "" ] || [ "$5" = "
    " ]; then
  echo "usage: $0 <length1> <length2> <step1> <step2> <output_dir>"
  exit 1
fi
```

```
7
8  output_dir=$5
9
10 length1=$1
11 length2=$2
12 step1=$3
13 step2=$4
14
15 count1=0
16 count2=$length1
17
18 while [ $count1 -lt $length1 ] || [ $count2 -lt $((length1+length2)) ]
19 do
20   echo "ids1 ""$count1"" ... ""$((count1+step1-1))"
21   echo "ids2 ""$count2"" ... ""$((count2+step2-1))"
22
23   sem -j+0 python generator.py $length1 $step1 $step2 $count1 $count2 "${
       output_dir}/out1_${count1}.csv" "${output_dir}/out2_${count2}.csv"
24
25   count1=$((count1 + step1))
26   if [ $count1 -gt $((length1-step1)) ]
27   then
28     step1=$((length1-count1))
29   fi
30
31   count2=$((count2 + step2))
32   if [ $count2 -gt $((length1+length2-step2)) ]
33   then
34     step2=$((length1+length2-count2))
35   fi
36
37   echo ""
38 done
39
40 sem --wait
```

Listing 3.5: parallel_generator.sh

Συνένωση των chunks

Στην φάση αυτή τα chunks, συνενώνονται σε δύο τελικά αρχεία μορφής csv, που θα αποτελούν τους δύο πίνακες του join. Τα αρχεία αυτά θα χρησιμοποιηθούν ως πηγή άντλησης δεδομένων για την εκτέλεση των αλγορίθμων συνένωσης στις μηχανές εκτέλεσης.

```
1  #!/bin/bash
2
3
4  if [ "$1" = "" ] || [ "$2" = "" ] || [ "$3" = "" ]; then
5    echo "usage: $0 <results_dir> <output_file1> <output_file2"
6    exit 1
7  fi
8
9  results_dir=$1
10 output_file1=$2
11 output_file2=$3
12 pwd=`pwd`
13 cd $results_dir || exit 1
```

```
14
15
16 ':>$output_file1'
17 ':>$output_file2'
18
19 count1=0
20 count2=0
21
22 table1() {
23   for file1 in out1*
24   do
25     echo "count1 = ""$count1"
26     cat $file1 >> $output_file1 && rm $file1
27     count1=$((count1 + 1))
28   done
29 }
30
31 table2() {
32   for file2 in out2*
33   do
34     echo "count2 = ""$count2"
35     cat $file2 >> $output_file2 && rm $file2
36     count2=$((count2 + 1))
37   done
38 }
39
40 table1 &
41 table2 &
42 wait
43
44 cd $pwd
```

Listing 3.6: chunks concatination

Δημιουργία Dataset για το bucket Join

Για την δημιουργία πινάκων δεδομένων κατάλληλων για το bucket Join, θα πρέπει τα αρχεία csv, να έχουν και μία τρίτη στήλη που να δείχνει τον αριθμό του κάδου που θα τοποθετηθούν. Ο τρόπος που επιτυγχάνεται αυτό είναι με τη χρήση awk γλώσσας και bash scripting, και φαίνεται στο ακόλουθο τμήμα κώδικα.

```
1  #!/bin/bash
2  if [ "$1" = "" ] || [ "$2" = "" ] ; then
3      echo "usage: $0 <directory> <buckets> "
4        exit 1
5  fi
6
7  pwd='pwd'
8  cd $1 || exit 1
9
10 for file in ldataset*
11 do
12   echo "$file"
13   ':>$1b$file'
14   awk 'BEGIN {i=0;} {print $1,$2,i;i=(i+1)%B}' FS=, OFS=, B=$2 $1$file > $1b$file
15
16 done
17
```

```
18 cd $pwd
```

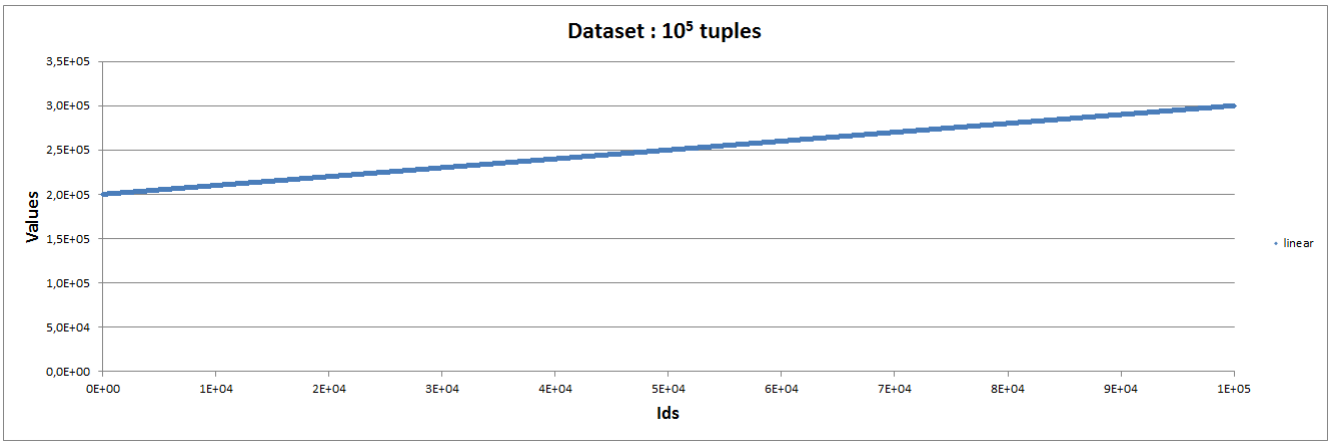
Listing 3.7: Create Bucket

3.4.2 Κατανομές

[18]

Linear

Η κατανομή αυτή είναι ουσιαστικά ”1-1” αντιστοιχία ανάμεσα σε id και value.



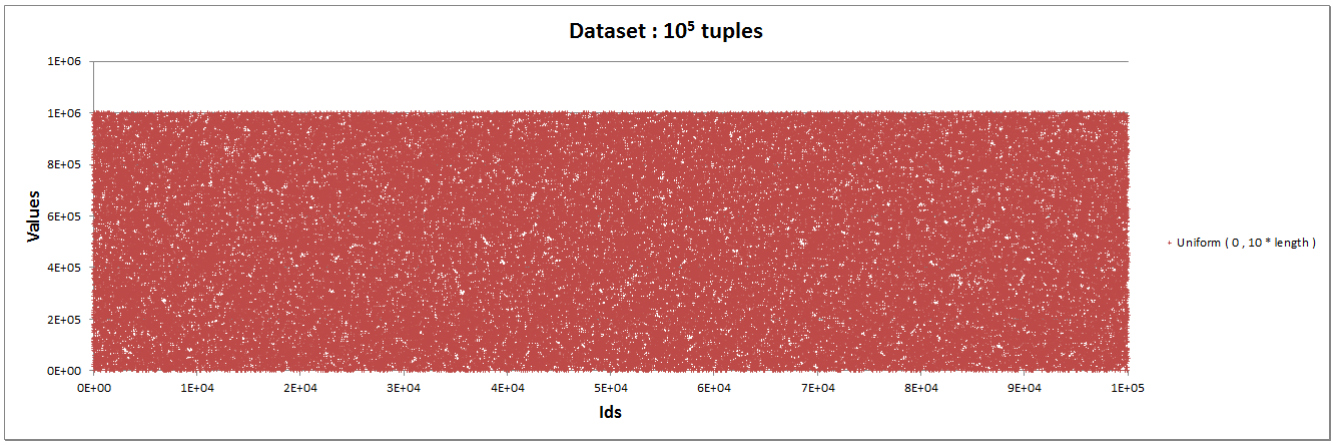
Σχήμα 3.6: Κατανομή Linear για dataset μεγέθους 10^5 πλειάδων

Uniform [4]

Η κατανομή έχει συνάρτηση πυκνότητας πιθανότητας :

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{else} \end{cases}$$

Όπως διαπιστώνει κανείς και από το ακόλουθο διάγραμμα, οι τιμές των values είναι ομοιόμορφα κατανεμημένες στα διάφορα ids.



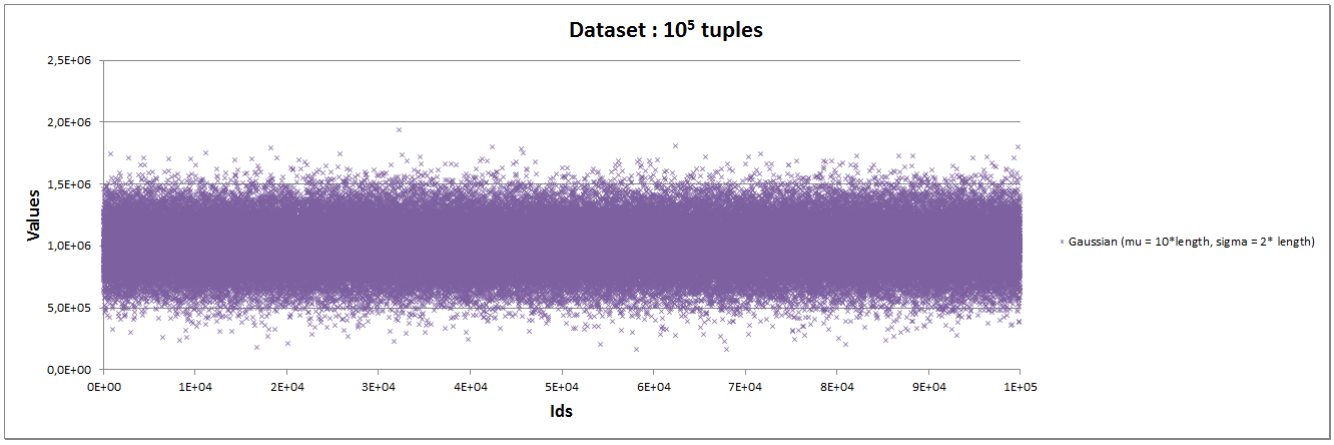
Σχήμα 3.7: Κατανομή Uniform για dataset μεγέθους 10^5 πλειάδων με a=0 και b=10^6

Gaussian [3]

Η κατανομή έχει συνάρτηση πυκνότητας πιθανότητας :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Παρατηρώντας κανείς το ακόλουθο διάγραμμα μπορεί να διαπιστώσει ότι υπάρχει μεγάλο πλήθος από ids που έχουν value κοντά στην μέση τιμή (10^6) και μικρότερο όσο απομακρύνονται οι τιμές του value από αυτήν. Η ”καμπάνα” της κανονικής κατανομής είναι εμφανής στον κάθετο άξονα.



Σχήμα 3.8: Κατανομή Gaussian (Normal) για dataset μεγέθους 10^5 πλειάδων

Οι κατανομές επιλέχτηκαν για τον καθορισμό του data skew. Δεδομένης ωστόσο της τυχαιότητας, προκύπτει το γεγονός ότι το μέγεθος του αποτελέσματος του join των δύο πινάκων, δεν μπορεί να καθοριστεί ντετερμινιστικά. Για παράδειγμα στην κανονική κατανομή, υπάρχουν πολλά ids (έστω x) που έχουν για value την μέση τιμή. Έτσι, εαν ο μικρός πίνακας περιέχει αυτή την τιμή, τότε αντί για 1 πλειάδα στην έξοδο, θα υπάρχουν x. Δεδομένου ότι το x είναι μία τυχαία μεταβλητή, δεν είναι δυνατόν να καθοριστεί το τελικό μέγεθος της εξόδου.

Προς την κατεύθυνση αυτή, επιλέχθηκε και η ντετερμινιστική Linear κατανομή, με σκοπό την παραγωγή μοντέλων όπου το τελικό μέγεθος του πίνακα εξόδου πρέπει να μπορεί να ελεγχθεί με ακρίβεια.

Κεφάλαιο 4

Υλοποίηση του Συστήματος και Αξιολόγηση των αποτελεσμάτων

Σε αυτό το κεφάλαιο παρουσιάζεται η υλοποίηση του συστήματος και πιο συγκεκριμένα η παρουσίαση των σημαντικότερων αποτελεσμάτων της διαδικασίας μοντελοποίησης. Συμπεριλαμβάνεται αξιολόγηση και ερμηνεία κάθε αξιοσημείωτου αποτελέσματος.

Αρχικά στο Τμήμα 4.1 δίνεται περιληπτικά η δομή των παραγόμενων μοντέλων καθώς και λεπτομέρειες για τον τρόπο πραγματοποίησης της διαδικασίας μοντελοποίησης. Ακολουθεί παρουσίαση και αναλυτική περιγραφή των αποτελεσμάτων για κάθε έναν από τους αλγορίθμους συνένωσης που μελετούνται. Το Τμήμα 4.2 σχετίζεται με τον αλγόριθμο reduce-side join (2.1.1). Το Τμήμα 4.3 σχετίζεται με τον αλγόριθμο map-side broadcast join (2.1.2.1). Τέλος το Τμήμα 4.4 σχετίζεται με τον αλγόριθμο map-side bucket join (2.1.2.2).

4.1 Δομή και υλοποίηση Μοντέλων

Για κάθε έναν από τους 3 αλγορίθμους συνένωσης που μελετώνται, πραγματοποιείται η διαδικασία μοντελοποίησης. Πίο συγκεκριμένα η πραγματοποίηση της κάθε φάσης της διαδικασίας (3.2) έχει τα εξής χαρακτηριστικά:

1. **Εκτέλεση πειραμάτων** (Run Experiments) :

   - Χρησιμοποιείται ο generator 3.4.1 για την δημιουργία όλων των σετ δεδομένων και αποθήκευσή του σε ένα volume μεγέθους 1TB που είναι προσαρτημένο στο κόμβο master. Το volume αυτό λειτουργεί ως χώρος άντλησης δεδομένων (data pool) για τις εκτελέσεις.
   - Τα πειράματα εκτελούνται με scripts αντίστοιχα του ζητούμενου αλγορίθμου και της ζητούμενης μηχανής εκτέλεσης.
   - Οι μηχανές εκτέλεσης των οποίων μοντελοποιείται η επίδοσή είναι:
     - *Postgresql* 2.4.1
     - *Hive* 2.4.3
     - *Spark (on YARN)* 2.4.4
   - Οι παράμετροι που μεταβάλλονται στις διάφορες εκτελέσεις πειραμάτων αναλυτικά για τις 3 κατηγορίες είναι [1]:

[1] Αξίζει να σημειωθεί, ότι οι παράμετροι αυτοί δίνονται εν μέρη από τον χρήστη. Πιο συγκεκριμένα, το Output Table Size και το Data Skew δεν δίνονται ως είσοδο. Πιθανόν να μπορούν να εξαχθούν σε ένα ενδιάμεσο επίπεδο από τον Planner.

(a) Παράμετροι συνόλων δεδομένων (**Data Specific Parameters**):
   - *Μέγεθος μεγάλου πίνακα συνένωσης (Big Table Size)*
   - *Μέγεθος μικρού πίνακα συνένωσης (Small Table Size)*
   - *Μέγεθος πίνακα εξόδου (Output Table Size)* [2]
   - *Κατανομή δεδομένων στους πίνακες (Data Skew)*

(b) Παράμετροι πόρων (**Resource Specific Parameters**):
   - *Αριθμός κόμβων (Number of nodes)*
   - *Μέγεθος πόρων ανά κόμβο (Resources per node)*
     * RAM Memory size
     * CPU number of cores
     * Root Disk size

(c) Παράμετροι αλγορίθμων (**Algorithm Specific Parameters**)
   - *Αριθμός πινάκων συνένωσης (Number of Join Tables)* [3]
   - *Αριθμός κάδων στο map - side Bucket Join (Number of buckets)*

- Τα αποτελέσματα σχετικά με την επίδοση των πειραμάτων, παρατηρήθηκε ότι εμφανίζουν διακύμανση στις τιμές τους (θόρυβο) για διάφορες εκτελέσεις. Αυτό οφείλεται στην υποδομή του openstack, και συγκεκριμένα στα εξής:
  - Οι εικονικές μηχανές παρουσιάζουν ετερογένεια στους υπολογιστικούς πόρους τους.
  - Το επίπεδο της επίδοσης, είναι απομακρυσμένο από το φυσικό επίπεδο όπου τρέχουν τα πειράματα, καθώς οι μηχανές είναι εικονικές
  - Υπάρχει ένα αρκετά μεγάλος (για το μέγεθος των πόρων του openstack) αριθμός χρηστών, με έναν άγνωστο αριθμό από άλλες εφαρμογές που τρέχουν παράλληλα στους ίδιους φυσικούς πόρους (δίκτυο, επεξεργαστές, δίσκους, μνήμη κτλ.)

  Έγινε προσπάθεια μείωσης κατά το δυνατόν περισσότερο του θορύβου με διαφόρους τρόπους:
  - Τα πειράματα που εκτελούνται για την Postgresql εκτελούνται με τις διάφορες διεργασίες των κατανεμημένων μηχανών (Hadoop,Hive,Spark) να έχουν διακοπεί.
  - Η εκτέλεση των πειραμάτων γίνεται σειριακά και συνεχόμενα.
  - Προτιμούνται ώρες όπου η χρησιμοποίηση του openstack είναι σχετικά χαμηλή.
  - Πολλαπλές μετρήσεις για εμφανώς θορυβώδεις μετρήσεις.

2. **Ανάκτηση Μετρήσεων** (Get / Parse Data) :

- Οι παράμετροι εξόδου που μετρούνται είναι:
  - *Χρόνος Εκτέλεσης* (Total Time)
  - *Χρησιμοποίηση CPU* (CPU Usage)
  - *Όγκος δεδομένων εισόδου - εξόδου* (I/O Bytes)
  - *Χρησιμοποίηση μνήμης RAM* (Memory Usage)

[2] Καθορίζεται από το ποσοστό ταιριάσματος κλειδιού των δύο πινάκων και τίθεται κατά την δημιουργία των σετ δεδομένων. Έχει ως αποτέλεσμα το μέγεθος του πίνακα εξόδου να λαμβάνει τιμές στο διάστημα (0, Μέγεθος Μικρού Πίνακα]. Στα αποτελέσματα που παρουσιάζονται είναι στο 100%.

[3] Είναι σταθερός και ίσος με 2 στην παρούσα μελέτη.

- Ο χρόνος εκτέλεσης μετράται απευθείας από την έξοδο που δίνουν οι ίδιες οι μηχανές εκτέλεσης.
- Η χρησιμοποίηση CPU και ο όγκος δεδομένων εισόδου - εξόδου μετρούνται με τη βοήθεια του ganglia 2.3.1 από το οποίο γίνεται εξαγωγή των μέσων όρων όλων των κόμβων για το χρονικό διάστημα που διήρκεσε η εκτέλεση του πειράματος.
- Η χρησιμοποίηση μνήμης μετράται με τη βοήθεια του script 2.3.2.
- Τα δεδομένα εισάγονται και οργανώνονται σε πρόγραμμα τύπου spreadsheet.

3. **Εκπαίδευση των μοντέλων του Weka** (Train Weka Models) :

   - Τα αποτελέσματα των μετρήσεων εξάγονται σε μορφή csv από το spreadsheet πρόγραμμα.
   - Εισάγονται στο εργαλείο προ επεξεργασίας (preprocess tool) του Weka.
   - Εκεί γίνεται επιλογή των γραμμών (values) και των στηλών (attributes) που θα χρησιμοποιηθούν.
   - Χρησιμοποιείται το εργαλείο classification του Weka, προκειμένου να εκπαιδευτεί το υπό εξέταση μοντέλο.
   - Τα μοντέλα αποθηκεύονται σε μορφή που μπορούν να χρησιμοποιηθούν μελλοντικά για περαιτέρω εκπαίδευση αλλά και πρόβλεψη.

4. **Εκτίμηση σφαλμάτων πρόβλεψης** (Evaluate Prediction Errors) :

   - Χρησιμοποιείται το visualization εργαλείο του Weka σε συνδυασμό με τα αποτελέσματα των σφαλμάτων των προβλέψεων του υπό εξέταση μοντέλου.
   - Για τα διάφορα υπό εξέταση μοντέλα, γίνεται σύγκριση των:
     - *Μέσο Απόλυτο Σφάλμα (Mean absolute error)*
     - *Μέσο τετραγωνικό Σφάλμα (Root mean squared error)*
     - *Σχετικό Απόλυτο Σφάλμα (Relative absolute error)*
     - *Σχετικό τετραγωνικό Σφάλμα (Root relative squared error)*
     - *Επιμέρους σφαλμάτων για την κάθε μέτρηση*
   - Επιλέγεται το μοντέλο με τα μικρότερα σφάλματα. [4]

   Στο σημείο αυτό, αξίζει να σημειωθεί ότι σε κάποιες περιπτώσεις, εξ' αιτίας ανωμαλιών στη συμπεριφορά των μετρήσεων (π.χ. μικρή τάση μέχρι κάποια τιμή -> απότομη αύξηση -> διαφορετική τάση για τις υπόλοιπες μετρήσεις), έγινε επιλογή ξεχωριστών μοντέλων, ένα για κάθε σύνολο μετρήσεων με παρόμοια συμπεριφορά. Με αυτό τον τρόπο επιτεύχθηκε πολύ μεγάλη βελτίωση στο σφάλμα και έτσι εγκυρότερο τελικό μοντέλο.

5. **Δειγματοληψία (Sampling)** :
   Δεδομένου ότι οι παράμετροι εισόδου είναι πολλοί, είναι πρακτικά αδύνατον ο δειγματοχώρος να καλυφθεί εξ' ολοκλήρου. Έτσι πραγματοποιήθηκε μία επιλογή των δειγμάτων χρησιμοποιώντας εποπτικές τεχνικές, με αποτέλεσμα το κλάδεμα του συνόλου των μοντέλων (pruning).

   Πιο συγκεκριμένα, η μεταβαλλόμενη παράμετρος σε κάθε πείραμα μελετήθηκε για έναν συνδυασμό σταθερών τιμών των υπόλοιπων παραμέτρων, τέτοιες ώστε να συνοψίσει το

[4] MLP : Multilayer perceptron model

αποτέλεσμα και για τις υπόλοιπες. Δεδομένης μίας μέγιστης και μίας ελάχιστης τιμής της μεταβαλλόμενης παραμέτρου, λαμβάνονται στον πρώτο κύκλο της διαδικασίας μοντελοποίησης ένας σταθερός αριθμός μετρήσεων (που εξαρτάται από το εκάστοτε εύρος μέγιστης και ελάχιστης τιμής) τέτοιες ώστε να απέχουν μεταξύ τους συγκεκριμένη απόσταση (π.χ. δεκαπλάσια σχέση για τα μεγέθη των πινάκων). Στους επόμενους κύκλους της διαδικασίας εκτιμάται η συμπεριφορά του μοντέλου, και λαμβάνονται επιμέρους περισσότερες μετρήσεις στα διαστήματα όπου το μοντέλο παρουσιάζει απότομες αλλαγές με σκοπό να γίνει πιο ακριβές.

Τα ακόλουθα μοντέλα παρουσιάζονται με την μορφή δισδιάστατων γραφημάτων και αποτελούν τα σημαντικότερα από πλευράς ενδιαφέροντος αποτελέσματα της διπλωματικής αυτής. Οι μετρήσεις (εκτός από αυτές που είναι για μεταβαλλόμενες παραμέτρους πόρων) έχουν εξαχθεί από πειράματα σε σύμπλεγμα με τα εξής χαρακτηριστικά:

- 4 nodes
- 8GB RAM per node
- 4VCPUs per node
- 60GB root disk per node
- Κατανομή Dataset : Linear

4.2 Εκτέλεση με Reduce - Side Join

4.2.1 Υλοποίηση

```
#!/bin/bash

on_die()
{
  kill $(jobs -pr)
  exit 0
}

trap 'on_die' SIGINT SIGTERM

#linear

A=("8" "8" "8" "8" "8" "8" "8" "8" "8")
B=("2" "3" "4" "5" "6" "6" "6" "6" "6")
C=("" "" "" "" "" "2" "4" "6" "8")

for ((i=0;i<${#A[@]};i++)); do

  echo "${A[$i]}_${B[$i]}${C[$i]}"

#load tables

  hive -e "create table ldataset${A[$i]}_${B[$i]}${C[$i]}(id1 BIGINT, value1 BIGINT)
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
␣␣LOAD␣DATA␣LOCAL␣INPATH␣'/media/data/dataset/linear_dataset/ldataset${A[$i]}_${B[$i]}
    ${C[$i]}.csv'␣OVERWRITE␣INTO␣TABLE␣ldataset${A[$i]}_${B[$i]}${C[$i]};
␣␣create␣table␣ldataset${B[$i]}_${A[$i]}${C[$i]}(id2␣BIGINT,␣value2␣BIGINT)␣ROW␣FORMAT
    ␣DELIMITED␣FIELDS␣TERMINATED␣BY␣',';
␣␣LOAD␣DATA␣LOCAL␣INPATH␣'/media/data/dataset/linear_dataset/ldataset${B[$i]}_${A[$i]}
    ${C[$i]}.csv'␣OVERWRITE␣INTO␣TABLE␣ldataset${B[$i]}_${A[$i]}${C[$i]};

␣␣drop␣table␣hive_lout${A[$i]}_${B[$i]}${C[$i]};
␣␣drop␣table␣spark_lout${A[$i]}_${B[$i]}${C[$i]};"

#spark

  $SPARK_HOME/bin/spark-submit --master yarn spark_join.py -l ${A[$i]} ${B[$i]} ${C[$i
    ]}

#hive
  echo "EXECUTING␣ON␣HIVE"
  date
  mem_usage.sh &

  #reduce_side
  hive -e "CREATE␣TABLE␣hive_lout${A[$i]}_${B[$i]}${C[$i]}␣AS␣SELECT␣a.id1␣as␣id1,␣b.
    id2␣as␣id2␣,a.value1␣FROM␣ldataset${A[$i]}_${B[$i]}${C[$i]}␣a␣JOIN␣ldataset${B[$i
    ]}_${A[$i]}${C[$i]}␣b␣ON␣a.value1=b.value2;"

  date
  kill $(jobs -pr)
  wait #wait for mem_usage to finish

#drop tables
  hive -e "drop␣table␣hive_lout${A[$i]}_${B[$i]}${C[$i]};
␣␣drop␣table␣spark_lout${A[$i]}_${B[$i]}${C[$i]};"
  hive -e "drop␣table␣ldataset${A[$i]}_${B[$i]}${C[$i]};
␣␣drop␣table␣ldataset${B[$i]}_${A[$i]}${C[$i]};
␣␣drop␣table␣bldataset${A[$i]}_${B[$i]}${C[$i]};
␣␣␣␣␣␣␣␣drop␣table␣bldataset${B[$i]}_${A[$i]}${C[$i]};"

done
```

Listing 4.1: Script για την υλοποίηση με Reduce - Side Join σε **Spark** και **Hive**

Στο παραπάνω Script φαίνεται ο τρόπος με τον οποίο εκτελούνται τα πειράματα. Κάποιες βασικές παρατηρήσεις είναι:

- Τα σύνολα δεδομένων είναι αποθηκευμένα με τη μορφή: DdatasetA_BC , όπου
  - D : distribution (l -> linear | g -> gaussian | u -> uniform)
  - Big Table Size = $C * 10^A$ [5]
  - Small Table Size = $C * 10^B$
- Με ανάλογο τρόπο πραγματοποιούνται και τα πειράματα για τις άλλες δύο κατανομές που για λόγους οικονομίας χώρου δεν παρουσιάζονται.
- Η καταμέτρηση χρησιμοποίησης μνήμης RAM είναι μόνο για τον κόμβο master (αφού εκεί εκτελείται το script). Επιλέχθηκε να γίνει μόνο για το hive και για την postgresql,

[5] Στην περίπτωση που το C είναι κενό ("") λαμβάνεται ως 1.

αφού στο spark οι μετρήσεις δεν οδηγούσαν σε κάποιο αξιόλογο συμπέρασμα. Το γεγονός αυτό, οφείλεται στο ότι το spark δεν χρησιμοποιεί προφανώς επιπλέον μνήμη πέραν αυτής που δεσμεύει αρχικά στον κόμβο master. Αντίθετα το hive και η postgres εμφανίζουν διαφορά στην μνήμη κατά την εκτέλεση μίας εργασίας.

Εντελώς ανάλογος είναι και ο τρόπος εκτέλεσης σε postgresql. Ο κώδικας και οι μετρήσεις της postgresql είναι προφανώς κοινές και για τους 3 αλγορίθμους.

```
#!/bin/bash

on_die()
{
  kill $(jobs -pr)
  exit 0
}

trap 'on_die' SIGINT SIGTERM

#linear

A=("8" "8")
B=("6" "6")
C=("2" "4")

for ((i=0;i<${#A[@]};i++)); do

  echo "${A[$i]}_${B[$i]}${C[$i]}"

  psql -c "DROP TABLE ldataset${A[$i]}_${B[$i]}${C[$i]};"
  psql -c "CREATE TABLE ldataset${A[$i]}_${B[$i]}${C[$i]} (id BIGINT, value BIGINT) TABLESPACE linear;"

  psql -c "DROP TABLE ldataset${B[$i]}_${A[$i]}${C[$i]};"
  psql -c "CREATE TABLE ldataset${B[$i]}_${A[$i]}${C[$i]} (id BIGINT, value BIGINT) TABLESPACE linear;"


  psql -c "COPY ldataset${A[$i]}_${B[$i]}${C[$i]} FROM '/media/data/dataset/linear_dataset/ldataset${A[$i]}_${B[$i]}${C[$i]}.csv'  DELIMITER ',';"
  psql -c "COPY ldataset${B[$i]}_${A[$i]}${C[$i]} FROM '/media/data/dataset/linear_dataset/ldataset${B[$i]}_${A[$i]}${C[$i]}.csv'  DELIMITER ',';"


  date
  mem_usage.sh &


  psql -c "DROP TABLE post_ldataset${A[$i]}_${B[$i]}${C[$i]};"
  echo '\timing \\ CREATE TABLE post_ldataset'${A[$i]}_${B[$i]}${C[$i]}' AS SELECT a.id AS id1, b.id AS id2, a.value FROM ldataset'${A[$i]}_${B[$i]}${C[$i]}' AS a join ldataset'${B[$i]}_${A[$i]}${C[$i]}' AS b ON (a.value = b.value);' | LANG=C psql


  kill $(jobs -pr)
  wait

  date
```

```
44
45    psql -c "DROP␣TABLE␣ldataset${A[$i]}_${B[$i]}${C[$i]};"
46    psql -c "DROP␣TABLE␣ldataset${B[$i]}_${A[$i]}${C[$i]};"
47    psql -c "DROP␣TABLE␣post_ldataset${A[$i]}_${B[$i]}${C[$i]};"
48
49  done
```

Listing 4.2: Script για την υλοποίηση με Reduce - Side Join σε **Postgresql**

```
1  import sys, getopt
2  from pyspark.sql import HiveContext
3  from pyspark import SparkContext, SparkConf, StorageLevel
4
5  if len(sys.argv)<4:
6      print "Usage:␣join.py␣-g␣|␣-l␣|␣-u␣table1_size␣table2_size␣[point␣decade]"
7      quit()
8
9  t1 = sys.argv[2]
10 t2 = sys.argv[3]
11 if len(sys.argv)==5 :
12     t3 = sys.argv[4]
13 else:
14     t3 = ""
15
16 x = ""
17 try:
18     opts, args = getopt.getopt(sys.argv[1:],"glu")
19 except getopt.GetoptError:
20     print "Usage:␣join.py␣-g␣|␣-l␣|␣-u␣table1_size␣table2_size␣[point␣decade]"
21     sys.exit(2)
22
23 for opt, arg in opts:
24     if opt == '-g':
25         x = "g"
26     elif opt == '-u':
27         x = "u"
28     elif opt == '-l':
29         x = "l"
30
31 conf = SparkConf().setAppName(x+"out"+t1+"_"+t2+t3).setMaster("spark://master:7077")
32
33 sc = SparkContext(conf=conf)
34 sqlC = HiveContext(sc)
35
36 table1 = sqlC.table(x+"dataset"+t1+"_"+t2+t3).persist(StorageLevel.MEMORY_AND_DISK)
37
38 sqlC.sql("ANALYZE␣TABLE␣"+x+"dataset"+t2+"_"+t1+t3+"␣COMPUTE␣STATISTICS␣noscan")
39 table2 = sqlC.table(x+"dataset"+t2+"_"+t1+t3).persist(StorageLevel.MEMORY_AND_DISK)
40
41 res = table1.join(table2, table1.value1 == table2.value2)
42
43 res.registerTempTable("resH")
44 sqlC.sql("create␣table␣spark_"+x+"out"+t1+"_"+t2+t3+"␣as␣select␣id1,id2,value1␣from␣
       resH")
```

Listing 4.3: Python πρόγραμμα για την εκτέλεση του Join στο Spark

Στο παραπάνω πρόγραμμα φαίνεται ο τρόπος με τον οποίο εκτελούνται τα πειράματα στην μηχανή Spark. Κάποιες βασικές παρατηρήσεις είναι:

- Χρησιμοποιείται το HiveContext, προκειμένου να αποκτηθεί πρόσβαση στα δεδομένα που είναι αποθηκευμένα στο metastore του Hive.
- Δημιουργούνται δύο RDD, για τους δύο πίνακες που πρόκειται να συνενωθούν.
- Αφού πραγματοποιηθεί το join, δημιουργείται και αποθηκεύεται πίνακας στο Hive με το αποτέλεσμα. Αξίζει να σημειωθεί ότι το γεγονός αυτό, εκτός από ζητούμενο (προκειμένου να ελεγχθεί η ορθότητα του αποτελέσματος), είναι και απαραίτητο για την εκτέλεση του join. Αυτό συμβαίνει επειδή το spark έχει πολιτική οκνηρής αποτίμησης και δεν ξεκινάει την διαδικασία της συνένωσης εάν δεν χρειαστεί κάπου το αποτέλεσμά της.

4.2.2 Μοντέλα Επίδοσης

Σε κάθε διάγραμμα οι διακεκομμένες γραμμές δείχνουν τις πραγματικές τιμές των μετρούμενων μεγεθών και οι συμπαγείς των μοντέλων. Στο κάθε μοντέλο αναγράφεται σε παρένθεση ποιο μοντέλο του weka χρησιμοποιήθηκε.

4.2.2.1 Μεταβαλλόμενο μέγεθος τελικού πίνακα εξόδου

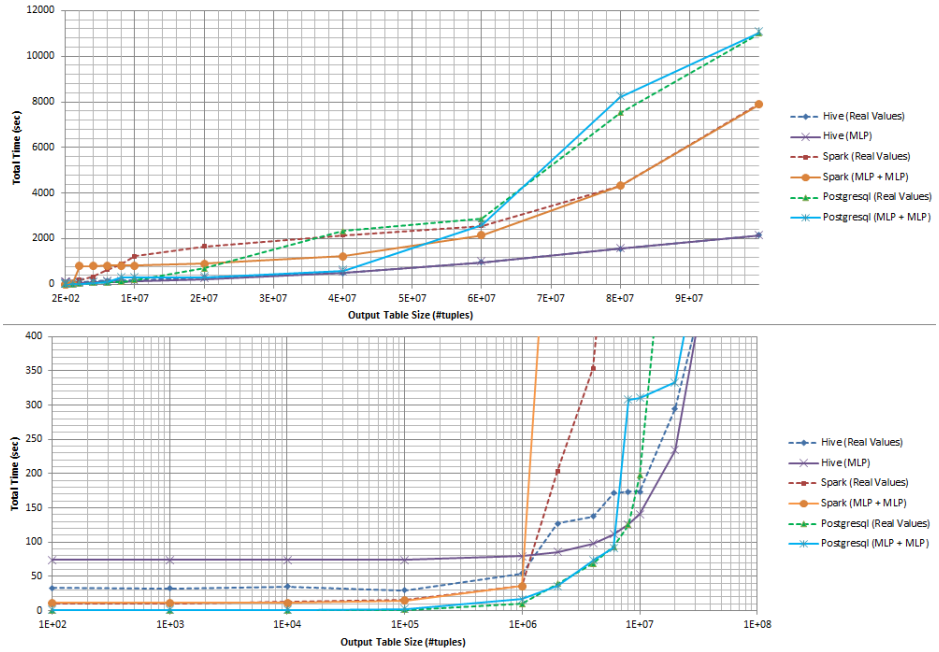
Στα πειράματα αυτά μεταβάλλεται το μέγεθος του μικρού πίνακα, που είναι ίσο με το μέγεθος του πίνακα εξόδου από 10^2 (4 KB) έως 10^8 (2.1 GB). Το μέγεθος του μεγάλου πίνακα είναι σε κάθε μέτρηση μία τάξη μεγέθους μεγαλύτερο από αυτό του μικρού δηλαδή από 10^3 (12 KB) έως 10^9 (20 GB). Έτσι φαίνεται η επίπτωση του πίνακα εξόδου στις διάφορες μετρικές.

Όπως διαπιστώνεται από το Σχ. 4.1, στο οποίο μετράται ο χρόνος εκτέλεσης, το Hive υπερτερεί σε επίδοση έναντι των άλλων δύο για μέγεθος εξόδου μεγαλύτερο από 10^7. Παρατηρώντας πιο προσεκτικά τα μοντέλα για μικρότερα μεγέθη προκύπτει ότι μέχρι και για μέγεθος 10^6 το spark έχει καλύτερη επίδοση από το hive, ενώ την καλύτερη την έχει η postgresql. Από την συμπεριφορά αυτή μπορούν να επιβεβαιωθούν κάποια χαρακτηριστικά των μηχανών εκτέλεσης.

Πιο συγκεκριμένα, η postgresql ως σχεσιακή βάση που εκτελείται σε έναν κόμβο, δεν έχει overhead λόγω κόστους επικοινωνίας μέσω δικτύου, αλλά και άλλες ανάγκες όπως συγχρονισμό, διαμοίραση πόρων και δεδομένων και συλλογή αποτελεσμάτων, που έχουν οι άλλες δύο κατανεμημένες μηχανές. Έτσι σε επίδοση είναι σαφέστατα προτιμότερη για σχετικά μικρά μεγέθη συνόλων δεδομένων.

Το spark και το hive, υπερτερούν σε επίδοση, σε μεγαλύτερα σύνολα δεδομένων. Αυτό είναι λογικό αφού το κύριο πλεονέκτημά τους, είναι η κλιμακωσιμότητα. Όσο αφορά την μεταξύ τους σύγκριση, το spark υπερτερεί του hive για σύνολα δεδομένων μικρότερα του 10^6.

Από τον πίνακα 4.1 μπορεί να παρατηρήσει κανείς αναλυτικά τους χρόνους και τα μεγέθη για τις 3 φάσεις που αποτελείται το reduce side join στο spark. Όπως μπορεί να διαπιστώσει κανείς, το μεγαλύτερο ποσοστό του χρόνου ξοδεύεται στην φάση του insert, γεγονός αναμενόμενο αφού η διαδικασία της τελικής συνένωσης στην reduce φάση είναι η πιο χρονοβόρα.



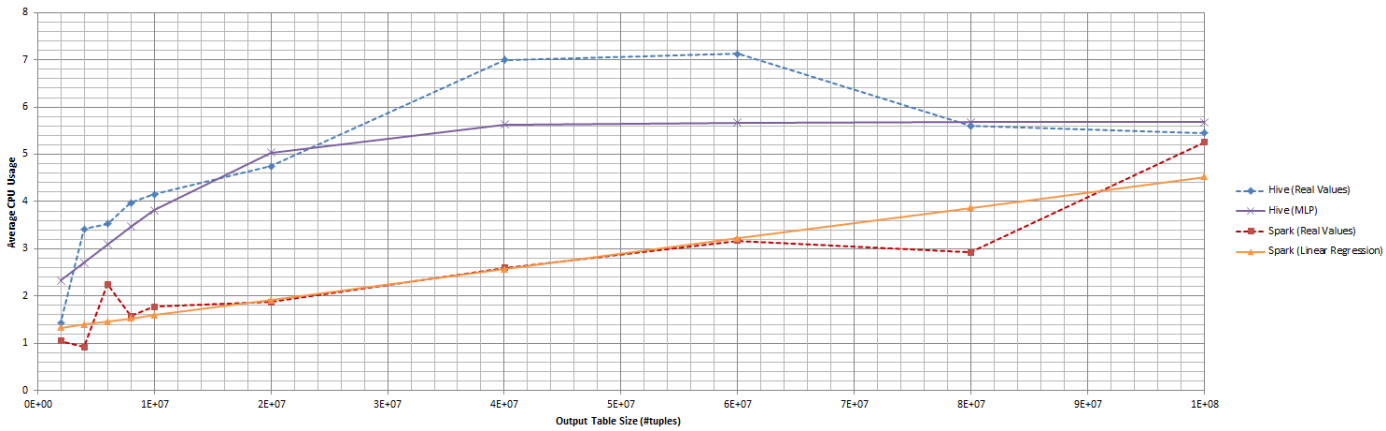
Σχήμα 4.1: Χρόνος Εκτέλεσης - Μέγεθος πίνακα εξόδου (reduce side)

Χωρίς μεγάλη διαφορά διάρκειας χρόνου ακολουθεί η map φάση του μεγάλου πίνακα. Το μέγεθος των εγγραφών shuffle όπως φαίνεται, είναι μία τάξης μεγέθους μεγαλύτερο από αυτό του μικρού. Εξού και η διαφορά στους χρόνους.

Map Big Table			Map Small Table			Insert		Total
Size	*Time (s)*	*Shuffle Write (Bytes)*	*Size*	*Time (s)*	*Shuffle Write (Bytes)*	*Time (s)*	*Shuffle Read (Bytes)*	*Time (s)*
10^3	5	66	10^2	5	16	6	82	11
10^4	5	644	10^3	5	67	5	710	11
10^5	6	6451	10^4	6	646	6	7066	13
10^6	8	62976	10^5	6	6349	4	69325	16
10^7	27	633344	10^6	9	63386	9	695910	36
$2*10^7$	96	1267405	$2*10^6$	24	126771	108	133837	204
$4*10^7$	324	2516582	$4*10^6$	17	253542	33	2831155	354
$6*10^7$	600	3879731	$6*10^6$	23	388506	34	4299162	660
$8*10^7$	840	5242880	$8*10^6$	96	522547	46	5767168	900
10^8	900	6501171	10^7	29	653005	354	7235174	1250
$2*10^8$	1200	13107200	$2*10^7$	840	1314304	474	14365491	1680
$4*10^8$	1554	26214400	$4*10^7$	1101	2621440	600	28835840	2167
$6*10^8$	1740	39426458	$6*10^7$	1320	3984589	840	43306189	2580
$8*10^8$	2640	52533658	$8*10^7$	2700	5242880	1500	57776538	4320
10^9	6120	65850573	10^8	3600	6606029	1980	72351744	7920

Πίνακας 4.1: Χαρακτηριστικά επίδοσης σε Spark

Για την χρησιμοποίηση της CPU δεν μετρήθηκαν οι επιδόσεις της postgresql. Ο λόγος για τον οποίο έγινε αυτό είναι γιατί κατά την εκτέλεση σε postgresql γίνεται υπολογιστική επιβάρυνση σε έναν μόνο κόμβο του συμπλέγματος. Αντίθετα στις κατανεμημένες μηχανές, χρησιμοποιούνται όλοι οι κόμβοι. Έτσι μία σύγκριση θα ήταν άνιση.



Σχήμα 4.2: Χρησιμοποίηση CPU - Μέγεθος μεγάλου πίνακα (reduce side)

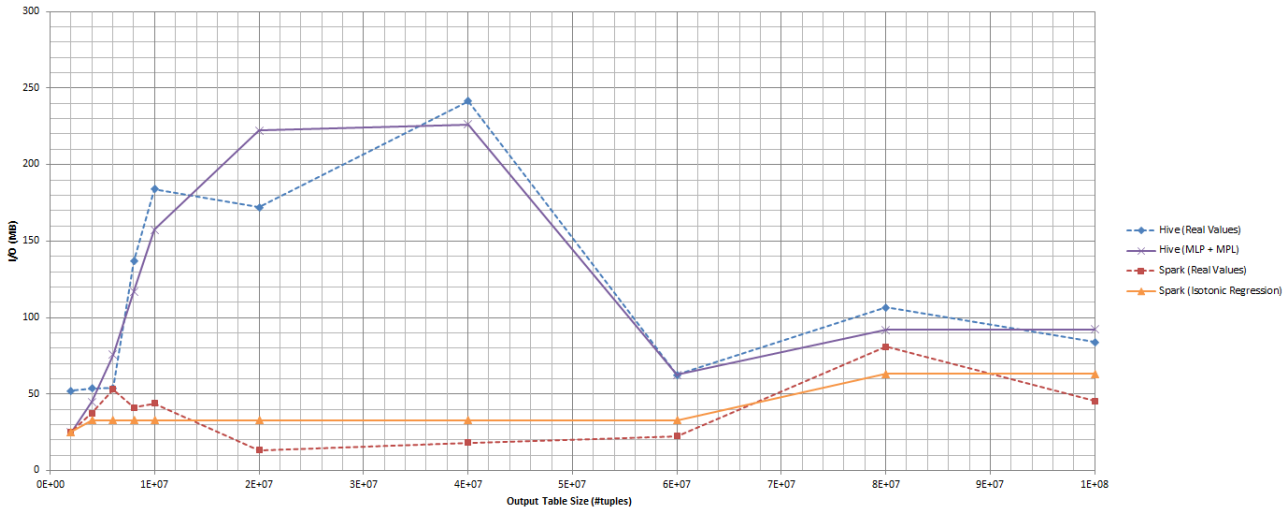
Όπως προκύπτει από το Σχ. 4.2 στο οποίο μετρική είναι η χρησιμοποίηση της CPU ως μέσος όρος στους κόμβους, το Spark εμφανίζει μικρότερη χρησιμοποίηση CPU από το hive. Ωστό-

σο το hive, παρουσιάζει μία φθίνουσα αυξητική τάση ενώ το spark σταθερή αυξητική. Στο hive, ο αριθμός των mappers και reducers αυξάνει καθώς αυξάνονται τα μεγέθη των πινάκων. Έτσι ο φόρτος κατανέμεται πιο ομαλά στους κόμβους με αποτέλεσμα η συνολική μέση χρησιμοποίηση CPU να παρουσιάζει αυτή την συμπεριφορά. Αντίθετα στο Spark, ο αριθμός των executors είναι δεδομένος και καθορίζεται από τους πόρους. Έτσι δικαιολογείται η σταθερή γραμμική αύξηση. Αξίζει να σημειωθεί επίσης, ότι εάν κάποιος χρήστης αποβλέπει στην ελαχιστοποίηση αυτής της παραμέτρου και μόνο, (π.χ. επειδή οι πόροι είναι διαμοιραζόμενοι και δεν πρέπει να υπερφορτωθούν) είναι προτιμότερο να εκτελεστούν οι δουλειές στην postgresql η οποία επιβαρύνει (σε όχι μεγαλύτερο ποσοστό από ότι οι άλλες μηχανές) μόνο έναν κόμβο αντί για όλους.

Η χρησιμοποίηση CPU προέκυψε από τον μέσο όρο της χρησιμοποίησης σε κάθε κόμβο του συμπλέγματος. Για να είναι έγκυρα τα δεδομένα, θεωρήθηκε σκόπιμο να παρθούν μετρήσεις μόνο για εργασία με χρόνο εκτέλεσης αρκετά μεγάλο (ώστε οι μετρήσεις να έχουν όσο δυνατόν λιγότερο θόρυβο γίνεται).

Οι μετρήσεις για την κατηγορία επίδοσης όσο αφορά τον όγκο δεδομένων εισόδου - εξόδου, στοχεύουν στην μοντελοποίηση της επιβάρυνσης του δικτύου των κόμβων κατά την εκτέλεση πιθανών εργασιών. Συνεπώς, και πάλι η postgresql δεν μπορεί να συγκριθεί καθώς δεν χρησιμοποιεί το δίκτυο δεδομένου ότι εκτελείται σε ένα κόμβο.

Στο Σχ. 4.3 μετράται η δικτυακή κίνηση στο σύμπλεγμα, ώς άθροισμα του συνολικού όγκου δεδομένων εξόδου και εισόδου για όλους τους κόμβους. Παρατηρείται ότι το Spark και πάλι έχει καλύτερη συμπεριφορά, όσο αφορά στη χρήση του δικτύου. Τα shuffles που κάνει στην map φάση, φαίνεται να είναι πιο αποδοτικά από αυτά του hive. Στο spark μπορεί κάποιος να παρατηρήσει ότι υπάρχει μία αύξηση στο μέγεθος $8 * 10^7$ η οποία πιθανόν οφείλεται στο γεγονός ότι πλέον οι πίνακες δεν χωρούν αποδοτικά στην μνήμη RAM των κόμβων και αναγκάζονται να γίνουν spilled στον δίσκο. Το γεγονός αυτό είναι αισθητό αλλά δεν επηρεάζει πάρα πολύ την επίδοση. Στο hive παρατηρείται μια επίσης ενδιαφέρουσα συμπεριφορά, καθώς φαίνεται ότι από κάποιο μέγεθος και πάνω η επίδοση βελτιώνεται σημαντικά. Ο αριθμός των mapper και reducer που εκτελείται για κάθε διαφορετικό μέγεθος εμφανίζει βέλτιστη συμπεριφορά στο μέγεθος $6 * 10^7$. Τα δεδομένα είναι λογικά κατανεμημένα στους κόμβους του hdfs με τέτοιο τρόπο, ώστε να μην είναι απαραίτητη μεγάλη ανταλλαγή δεδομένων στο δίκτυο.

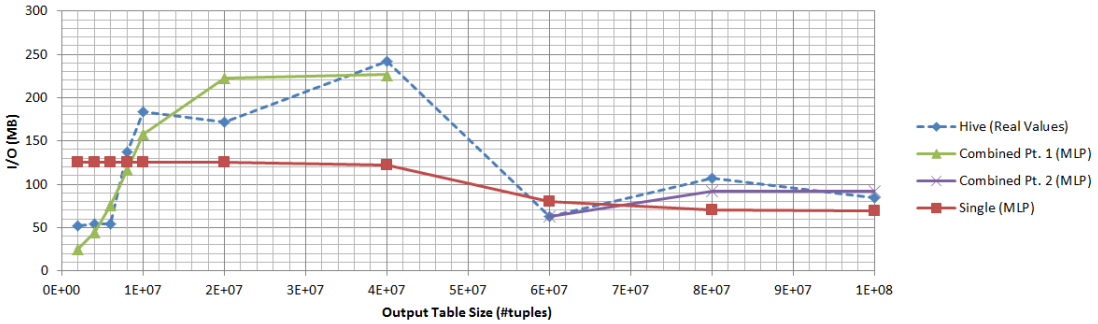


Σχήμα 4.3: Δικτυακή κίνηση - Μέγεθος μεγάλου πίνακα (reduce side)

Η καμπύλη του επίδοσης του Hive αποτελεί ένα χαρακτηριστικό παράδειγμα όπου χρησιμοποιήθηκαν δύο μοντέλα για την παραγωγή του τελικού. Ο λόγος είναι ότι υπάρχει μία αύξηση μέχρι μία συγκεκριμένη τιμή, ακολουθεί μια απότομη μείωση, και τέλος και πάλι μια αύξηση. Έτσι τα δεδομένα χωρίστηκαν σε δύο σετ με βάση την απότομη μείωση (Σχ. 4.4). Το τελικό μοντέλο είχε πολύ μικρότερα σφάλματα από το να χρησιμοποιούνταν ένα ενιαίο, όπως φαίνονται και στον Πίνακα 4.2. Τέλος αξίζει να σημειωθεί ότι η συμπεριφορά του hive, πιθανότατα οφείλεται στην αύξηση των mappers και reducers, με την αύξηση του μεγέθους. Η κατανομή τους στους 4 διαθέσιμους κόμβους, φαίνεται να λειτουργεί αποδοτικότερα, όσο αφορά την χρησιμοποίηση του δικτύου, για το μέγεθος $6 * 10^7$.

Error Type \Model Type	**Single Model**	**Combined Model Pt. 1**	**Combined Model Pt.2**
Correlation coefficient	0.2721	0.9341	0.8548
Mean absolute error	52.0863	24.7192	7.7026
Root mean squared error	61.2553	26.8444	9.7088
Relative absolute error	94.53%	38.72%	51.84%
Root relative squared error	97.16%	38.07%	53.76%

Πίνακας 4.2: Σύγκριση Μοντέλων

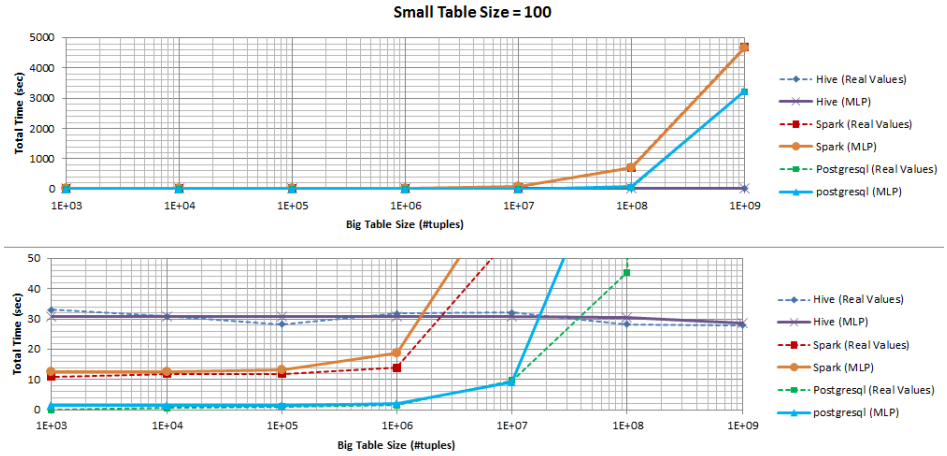


Σχήμα 4.4: Όγκος δεδομένων I/O - Μέγεθος μεγάλου πίνακα (reduce side) **Σύγκριση Μοντέλων**

4.2.2.2 Μεταβαλλόμενο μέγεθος μεγάλου πίνακα

Το μέγεθος του μικρού πίνακα παραμένει σταθερό και ίσο με 100 πλειάδες, ενώ το μέγεθος του μεγάλου πίνακα μεταβάλλεται από 10^3 εώς 10^9 πλειάδες.

Στο Σχ. 4.5 παρουσιάζονται τα μοντέλα για τον χρόνο εκτέλεσης. Ενδιαφέρον παρουσιάζει η επίδοση του Hive, η οποία φαίνεται να είναι ανεξάρτητη του μεγέθους του μεγάλου πίνακα. Αυτό έχει ως αποτέλεσμα από κάποιο μέγεθος και πάνω, το hive να αποτελεί την καλύτερη επιλογή. Τα optimizations που κρύβονται πίσω από το πλάνο εκτέλεσης των συγκεκριμένων αλγορίθμων, πιθανότατα οφείλονται για αυτή την συμπεριφορά.

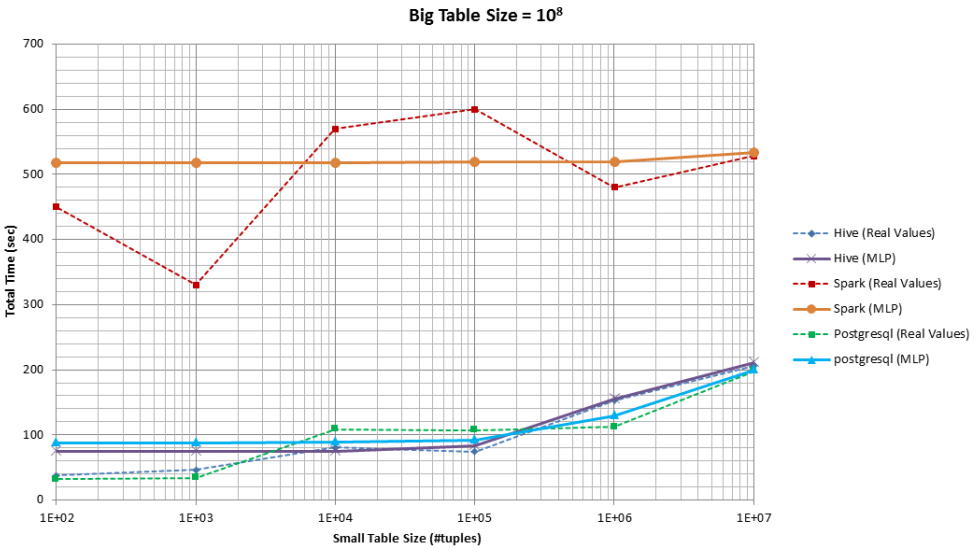


Σχήμα 4.5: Χρόνος Εκτέλεσης - Μέγεθος μεγάλου πίνακα (reduce side)

4.2.2.3 Μεταβαλλόμενο μέγεθος μικρού πίνακα

Το μέγεθος του μεγάλου πίνακα παραμένει σταθερό και ίσο με 10^8 ενώ το μέγεθος του μικρού πίνακα μεταβάλλεται.

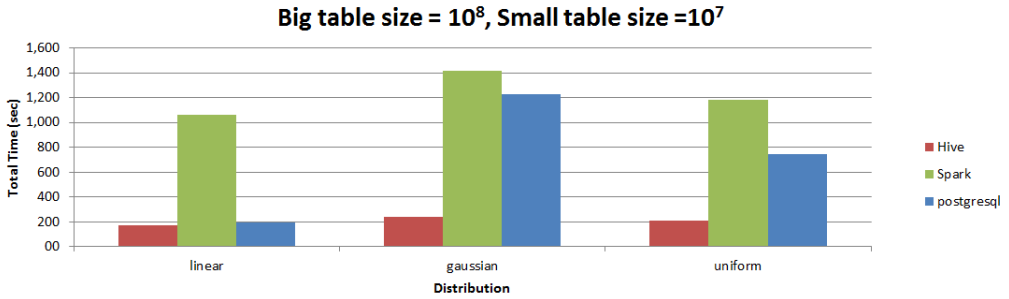
Στο Σχ. 4.6 δίνεται μια εικόνα για την επίπτωση στην επίδοση όσο αφορά τον χρόνο εκτέλεσης. Όπως φαίνεται η postgresql με το hive έχουν παρόμοια μοντέλα, ενώ το spark είναι σημαντικά πιο αργό και με επίδοση που φαίνεται ότι οφείλεται κυρίως στον επιπρόσθετο κόστος λόγω της κατασκευής in-memory καταστάσεως (RDD) και δρομολόγησης εργασιών με μη αποδοτικό (τουλάχιστον σε σύγκριση με το hive) τρόπο για τον μεγάλο πίνακα. Εξού και η σχετικά μικρή επίπτωση μεταβολής του μεγέθους του μικρού πίνακα. Η μεγάλη αυτή επίπτωση μπορεί να διαπιστωθεί και από τον πίνακα 4.1.



Σχήμα 4.6: Χρόνος Εκτέλεσης - Μέγεθος μικρού πίνακα (reduce side)

4.2.2.4 Μεταβαλλόμενη Κατανομή Συνόλου Δεδομένων

Σχετικά με την επιρροή της κατανομής που ακολουθούν τα δεδομένα, επιλέχθηκε να μην δημιουργηθούν μοντέλα, καθώς είναι πρακτικά ασύμφορο για τον χρήστη να γνωρίζει την κατανομή ενός τόσο μεγάλου όγκου δεδομένων. Στο παρακάτω διάγραμμα (Σχ. 4.7) παρουσιάζεται ωστόσο η επιρροή αυτή.



Σχήμα 4.7: Χρόνος Εκτέλεσης - Κατανομή (reduce side)

Η κατανομή των δεδομένων στους πίνακες, είναι αυτή που καθορίζει το μέγεθος του τελικού πίνακα μετά την συνένωση. Εν προκειμένω είναι φανερό ότι η κανονική κατανομή (gaussian) απαιτεί τον μεγαλύτερο χρόνο εκτέλεσης ενώ η γραμμική τον μικρότερο. Ωστόσο μπορεί να παρατηρήσει κανείς ότι οι διαφορές δεν είναι και τόσο μεγάλες, ενώ δεν φαίνεται να είναι ικανές να αλλάξουν την πρόβλεψη για την βέλτιστη επιλογή μηχανής εκτέλεσης.

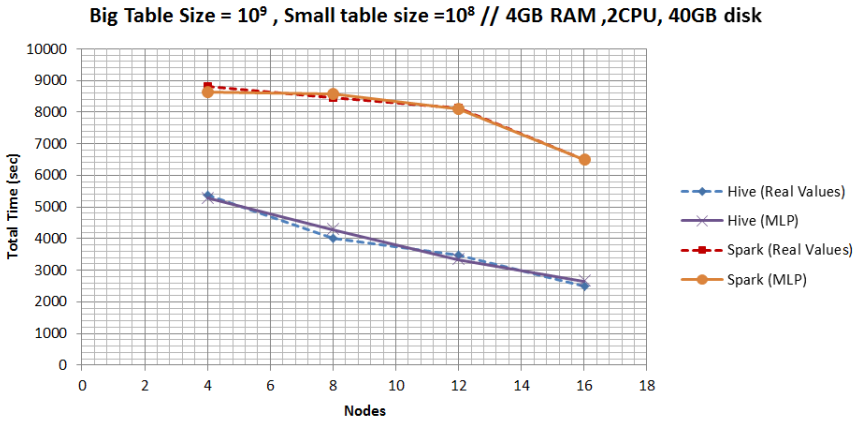
4.2.2.5 Μεταβαλλόμενος Αριθμός Κόμβων

Σε αυτή την κατηγορία, μεταβάλλεται ο αριθμός των κόμβων του συμπλέγματος με τους υπόλοιπους πόρους να παραμένουν σταθεροί:

- 4,8,12,16 nodes
- 4GB RAM per node
- 2VCPUs per node
- 40GB root disk per node
- Κατανομή Dataset : Linear

Η επίδραση της μεταβολής των κόμβων σχετικά με την επίδοση σε χρόνο εκτέλεσης, μελετήθηκε για τα μεγαλύτερα δυνατά σύνολα δεδομένων έτσι ώστε να δημιουργηθούν οι περισσότεροι δυνατοί σε αριθμό mapper και reducer. Όπως φαίνεται από το Σχ. 4.8 στο οποίο μετράται ο χρόνος εκτέλεσης σε σχέση με τον αριθμό των κόμβων υπάρχει μία σαφής βελτίωση και για τις δύο μηχανές, η οποία ωστόσο δεν είναι ιδιαίτερα μεγάλη. Φαίνεται λοιπόν ότι η επίδοση είναι φραγμένη από τους πόρους του κάθε κόμβου. Πιθανόν για μεγαλύτερα μεγέθη πόρων ανά κόμβο να ήταν πιο έντονη η βελτίωση με την αύξηση του αριθμού των κόμβων κυρίως για το spark που φαίνεται να μειώνει απότομα τον χρόνο εκτέλεσης στους 16 κόμβους. Επίσης το hive φαίνεται να έχει μεγαλύτερη μείωση στον χρόνο εκτέλεσης με την αύξηση των κόμβων, γεγονός που οφείλεται στην κατανομή των mapper και reducer σε περισσότερους κόμβους. Γενικό συμπέρασμα είναι ότι οι εφαρμογές αυτές είναι data intensive,

και για αυτό μεγαλύτερος αριθμός κόμβων φαίνεται να βελτιώνει, αλλά σε μικρό βαθμό την επίδοση.



Σχήμα 4.8: Χρόνος Εκτέλεσης - Αριθμός Κόμβων (reduce side)

4.3 Εκτέλεση με Map - Side Broadcast Join

4.3.1 Υλοποίηση

```
...

#spark

  $SPARK_HOME/bin/spark-submit --master yarn spark_join.py -l ${A[$i]} ${B[$i]} ${C[$i
    ]}

#hive
  echo "EXECUTING ON HIVE"
  date
  mem_usage.sh &

  #broadcast
  hive -d hive.ignore.mapjoin.hint=false -e "CREATE TABLE hive_lout${A[$i]}_${B[$i]}${
    C[$i]} AS SELECT /*+ MAPJOIN(b) */ a.id1 as id1, b.id2 as id2 , a.value1 FROM
    ldataset${A[$i]}_${B[$i]}${C[$i]} a JOIN ldataset${B[$i]}_${A[$i]}${C[$i]} b ON a.
    value1=b.value2;"

  date
  kill $(jobs -pr)
  wait #wait for mem_usage to finish

...
```

Listing 4.4: Script για την υλοποίηση με Map - Side Broadcast Join σε **Spark** και **Hive**

Στο Spark για την υλοποίηση του broadcast Join τέθηκε η τιμή στην παράμετρο *spark.sql.autoBroadcastJoinThreshold* ώστε να γίνεται αυτόματα broadcast για πίνακας μέσα σε αυτό το threshold. Στο hive η αντίστοιχη ρύθμιση είναι για την παράμετρο *HADOOP_CLIENT_OPTS*. Το μέγιστο μέγεθος πίνακα που μπορεί να χωρέσει στην cache

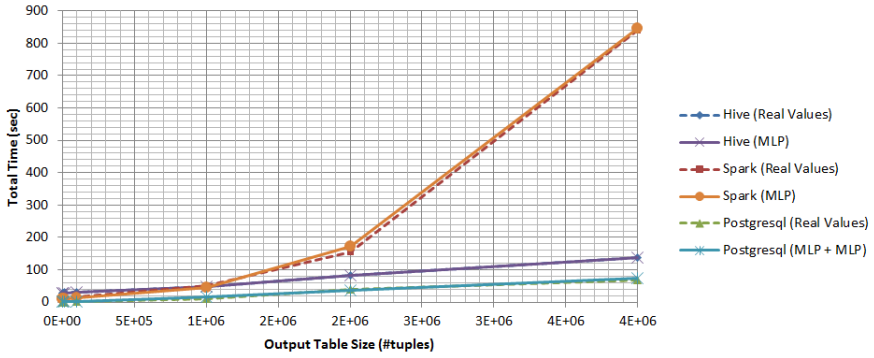
για τον χρησιμοποιούμενο συνδυασμό πόρων είναι $4 * 10^6$ (71 MB). Η τιμή αυτή προκύπτει αν αναλογιστεί κανείς την μνήμη που χρησιμοποιούν οι δαίμονες διεργασίες αλλά και την απαιτούμενη μνήμη για την εκτέλεση των εργασιών (Java heap size κτλ.)

4.3.2 Μοντέλα Επίδοσης

Τα μεγέθη των παραμέτρων παίρνουν τις ίδιες τιμές με του reduce - side με τη διαφορά ότι ο μικρός πίνακας έχει πλέον ως μέγιστη τιμή την $4 * 10^6$.

Από τα παρακάτω διαγράμματα, προκύπτει ότι γενικά ότι το spark είναι χειρότερο σε επίδοση από το hive όσο αφορά στον χρόνο. Φαίνεται ότι τα optimizations του hive για το map join είναι καλύτερα από αυτά του spark τουλάχιστον στα δεδομένα χαρακτηριστικά πόρων.

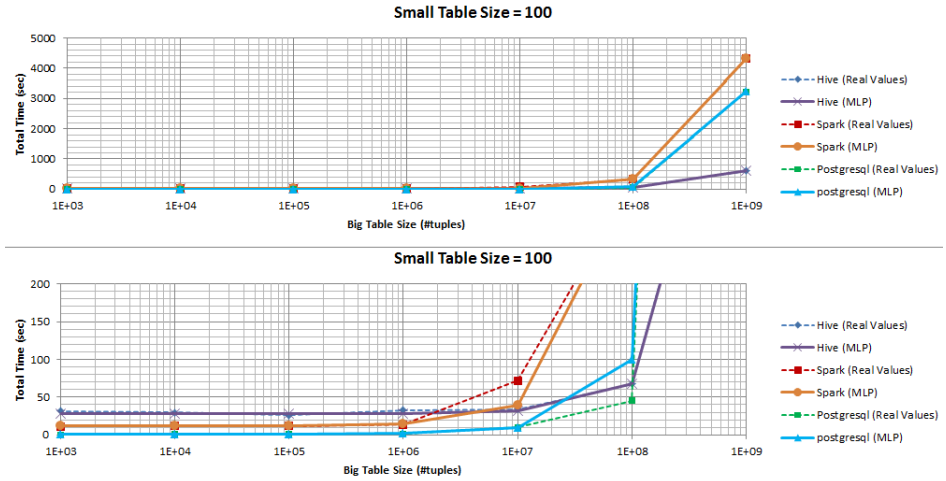
4.3.2.1 Μεταβαλλόμενο μέγεθος τελικού πίνακα εξόδου



Σχήμα 4.9: Χρόνος Εκτέλεσης - Μέγεθος πίνακα εξόδου (Map - side Broadcast Join)

4.3.2.2 Μεταβαλλόμενο μέγεθος μεγάλου πίνακα

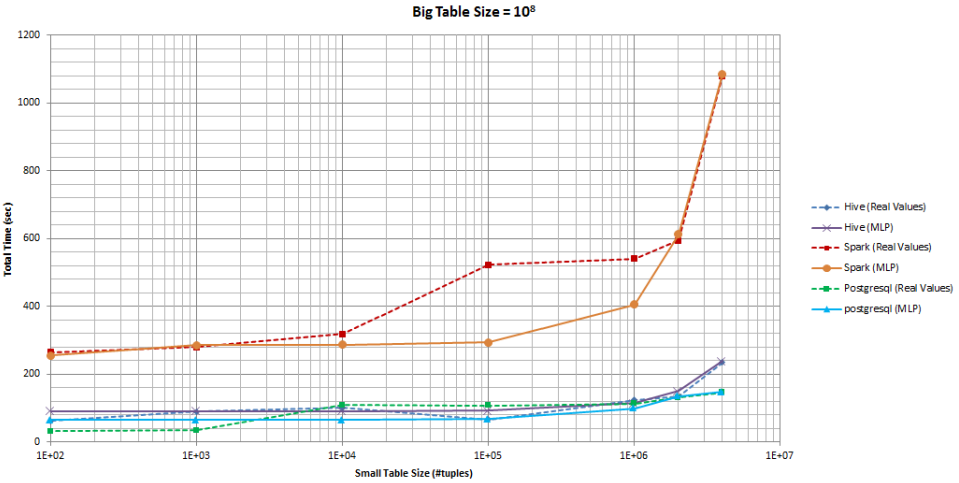
Γενικά η postgresql και το hive έχουν παρόμοια επίδοση. Ωστόσο στο μεγάλο μέγεθος πίνακα η postgresql είναι σημαντικά χειρότερη από το hive. Εκεί επιβεβαιώνεται το γεγονός ότι μπορεί η postgresql ως σχεσιακή βάση να είναι καλύτερη για μικρά μεγέθη, αλλά το βασικό της μειονέκτημα, που αποτελεί πλεονέκτημα για τις κατανεμημένες μηχανές εκτέλεσης, είναι ότι δεν μπορεί να κλιμακώσει καλά και από κάποιο μέγεθος και πάνω θα είναι πρακτικά αδύνατον ακόμα και να ολοκληρώσει την συνένωση.



Σχήμα 4.10: Χρόνος Εκτέλεσης - Μέγεθος μεγάλου πίνακα (Map - side Broadcast Join)

4.3.2.3 Μεταβαλλόμενο μέγεθος μικρού πίνακα

Παρατηρείται μία απότομη αύξηση στα μεγάλα μεγέθη, η οποία είναι πιο έντονη στο spark. Ο πίνακας, καθώς αποθηκεύεται στην μνήμη, και ειδικά στα μεγάλα μεγέθη που χωράει οριακά, μικραίνει τα περιθώρια για βελτιστοποιήσεις που βασίζονται στην χρησιμοποίηση μνήμης. Έτσι μπορεί να δικαιολογηθεί μια τέτοια συμπεριφορά.

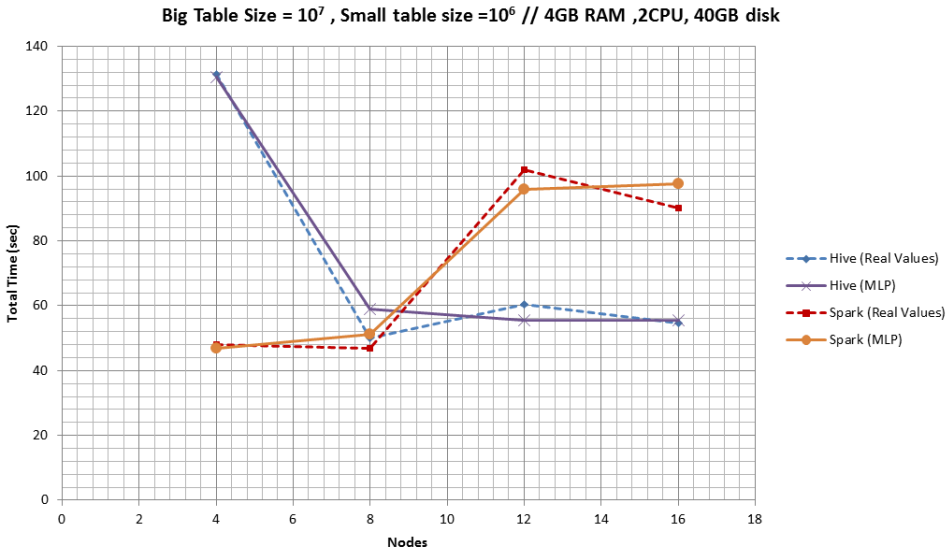


Σχήμα 4.11: Χρόνος Εκτέλεσης - Μέγεθος μικρού πίνακα (Map - side Broadcast Join)

4.3.2.4 Μεταβαλλόμενος Αριθμός Κόμβων

Ενδιαφέρον παρουσιάζουν τα αποτελέσματα σε μεταβαλλόμενο αριθμό κόμβων. Παρατηρείται ότι το hive βελτιώνει την απόδοσή του με την αύξηση του αριθμού των κόμβων ενώ αντίθετα το spark την χειροτερεύει. Ο πίνακας που γίνεται broadcasted σε κάθε mapper λογικά θα πρέπει να έχει μεγαλύτερο κόστος όσο ο αριθμός των κόμβων αυξάνεται. Αυτό είναι φανερό στο spark. Το hive αντίθετα φαίνεται ότι καταφέρνει με τις βελτιστοποιήσεις πλάνου εκτέλεσης για broadcast join να επωφεληθεί των περισσότερων κόμβων. Επίσης αξίζει να σημειωθεί ότι ο αριθμός των mappers είναι μικρότερος από τον αριθμό των κόμβων λόγω

μεγέθους των πινάκων. Έτσι δεν μπορεί να γίνει πλήρης εκμετάλλευση των περισσότερων κόμβων και για αυτό δεν παρατηρείται σαφής βελτίωση της επίδοσης.



Σχήμα 4.12: Χρόνος Εκτέλεσης - Αριθμός Κόμβων (Map - side Broadcast Join)

4.4 Εκτέλεση με Map - Side Bucket Join

4.4.1 Υλοποίηση

Για το Map - Side Bucket Join δεν υπάρχει υλοποίηση (επίσημη) στο Spark. Επιλέχθηκε να μην χρησιμοποιηθεί η υλοποίηση με Hive on Spark, καθώς ζητούμενο είναι η επίδοση των μηχανών ξεχωριστά, και όχι σε συνδυασμό. Για αυτό και μοντελοποιείται η επίδοση μόνο σε Hive.

```
...

#bucket load tables
  hive -e "set␣hive.enforce.bucketing=true;
␣␣set␣hive.exec.dynamic.partition.mode=nonstrict;

␣␣create␣table␣ldataset${A[$i]}_${B[$i]}${C[$i]}(id1␣BIGINT,␣value1␣BIGINT,␣bucket␣INT
    )␣␣ROW␣FORMAT␣DELIMITED␣FIELDS␣TERMINATED␣BY␣',’␣COLLECTION␣ITEMS␣TERMINATED␣BY␣
    '|'␣MAP␣KEYS␣TERMINATED␣BY␣':'␣;

␣␣LOAD␣DATA␣LOCAL␣INPATH␣'/media/data/dataset/linear_dataset/bldataset${A[$i]}_${B[$i
    ]}${C[$i]}.csv'␣OVERWRITE␣INTO␣TABLE␣ldataset${A[$i]}_${B[$i]}${C[$i]};

␣␣CREATE␣TABLE␣bldataset${A[$i]}_${B[$i]}${C[$i]}(id1␣BIGINT,value1␣BIGINT)␣
    PARTITIONED␣BY␣(bucket␣INT)␣CLUSTERED␣BY␣(value1)␣INTO␣10␣BUCKETS;

␣␣INSERT␣OVERWRITE␣TABLE␣bldataset${A[$i]}_${B[$i]}${C[$i]}␣PARTITION␣(bucket)␣select␣
    id1,value1,bucket␣from␣ldataset${A[$i]}_${B[$i]}${C[$i]};"

  hive -e "set␣hive.enforce.bucketing=true;␣␣␣set␣hive.exec.dynamic.partition.mode=
    nonstrict;

```

```
␣␣create␣table␣ldataset${B[$i]}_${A[$i]}${C[$i]}(id2␣BIGINT,␣value2␣BIGINT,␣bucket␣INT
    )␣␣ROW␣FORMAT␣DELIMITED␣FIELDS␣TERMINATED␣BY␣',␣COLLECTION␣ITEMS␣TERMINATED␣BY␣
    '|'␣MAP␣KEYS␣TERMINATED␣BY␣':'␣;

␣␣LOAD␣DATA␣LOCAL␣INPATH␣'/media/data/dataset/linear_dataset/bldataset${B[$i]}_${A[$i
    ]}${C[$i]}.csv'␣OVERWRITE␣INTO␣TABLE␣ldataset${B[$i]}_${A[$i]}${C[$i]};

␣␣CREATE␣TABLE␣bldataset${B[$i]}_${A[$i]}${C[$i]}(id2␣BIGINT,value2␣BIGINT)␣
    PARTITIONED␣BY␣(bucket␣INT)␣CLUSTERED␣BY␣(value2)␣INTO␣10␣BUCKETS;

␣␣INSERT␣OVERWRITE␣TABLE␣bldataset${B[$i]}_${A[$i]}${C[$i]}␣PARTITION␣(bucket)␣select␣
    id2,value2,bucket␣from␣ldataset${B[$i]}_${A[$i]}${C[$i]};"

#hive

  date
  #bucket

  hive -e "␣set␣hive.ignore.mapjoin.hint=false;␣␣set␣hive.optimize.bucketmapjoin=true;
    ␣CREATE␣TABLE␣hive_lout${A[$i]}_${B[$i]}${C[$i]}␣AS␣SELECT␣/*+␣MAPJOIN(b)␣*/␣a.id1
    ␣as␣id1,␣b.id2␣as␣id2␣,a.value1␣FROM␣bldataset${A[$i]}_${B[$i]}${C[$i]}␣a␣JOIN␣
    bldataset${B[$i]}_${A[$i]}${C[$i]}␣b␣ON␣a.value1=b.value2;"

...
```

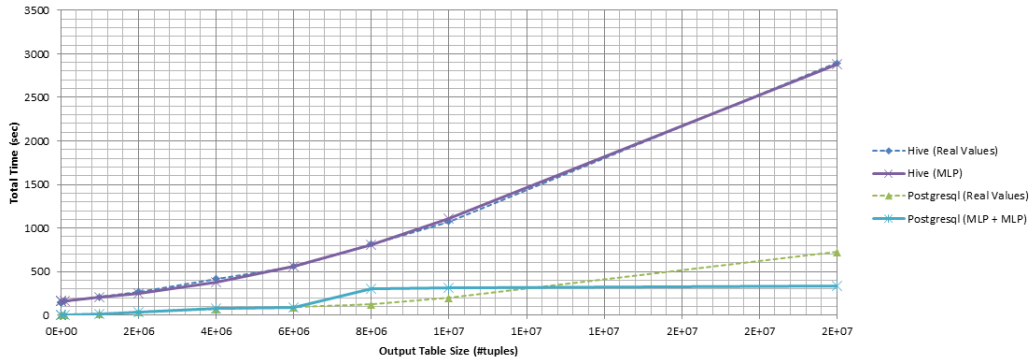
Listing 4.5: Script για την υλοποίηση με Map - Side Bucket Join σε **Spark** και **Hive**

4.4.2 Μοντέλα Επίδοσης

Ο Αριθμός των bucket έχει τεθεί ίσος με 10 έτσι ώστε να μπορέσει να γίνει η συνένωση για πίνακα μίας τάξης μεγέθους μεγαλύτερο από αυτόν του broadcast. Έτσι οι τιμές των παραμέτρων στα μοντέλα είναι όμοιες με τη διαφορά, ότι μέγεθος του μικρού πίνακα φτάνει πλέον την τιμή $4 * 10^7$ (783 MB).

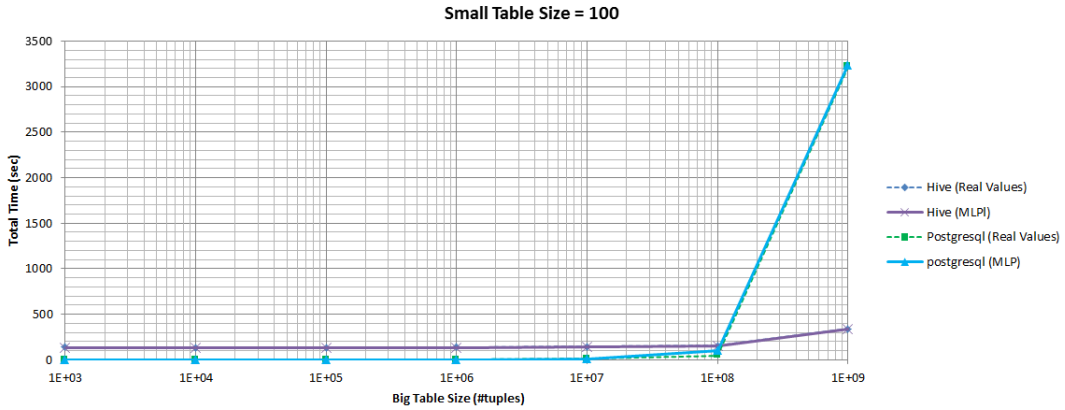
Είναι εμφανές ότι ο αλγόριθμος αυτός δίνει στο hive σαφώς χειρότερη επίδοση από την postgresql. Η συμπεριφορά του σε σχέση με τις διακυμάνσεις στις καμπύλες, είναι παρόμοια με αυτή του broadcast join, δεδομένου ότι και οι δύο αλγόριθμοι ανήκουν στην οικογένεια του map - side join και έχουν κοινά χαρακτηριστικά στον τρόπο λειτουργίας.

4.4.2.1 Μεταβαλλόμενο μέγεθος τελικού πίνακα εξόδου



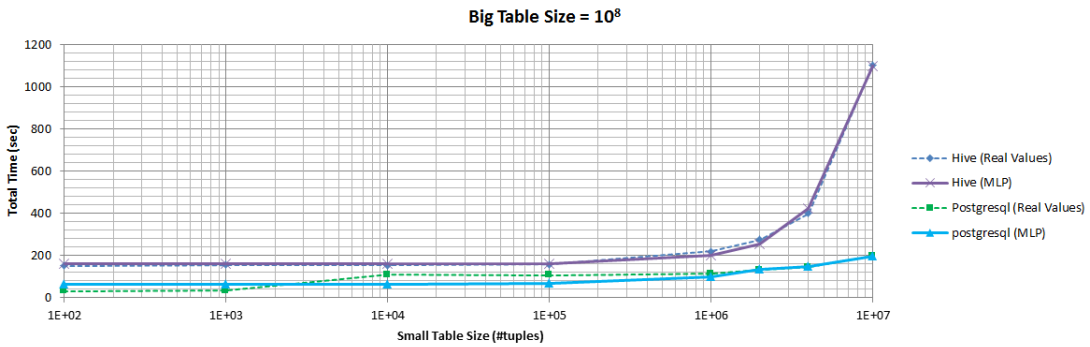
Σχήμα 4.13: Χρόνος Εκτέλεσης - Μέγεθος πίνακα εξόδου (Map - side Bucket Join)

4.4.2.2 Μεταβαλλόμενο μέγεθος μεγάλου πίνακα



Σχήμα 4.14: Χρόνος Εκτέλεσης - Μέγεθος μεγάλου πίνακα (Map - side Bucket Join)

4.4.2.3 Μεταβαλλόμενο μέγεθος μικρού πίνακα

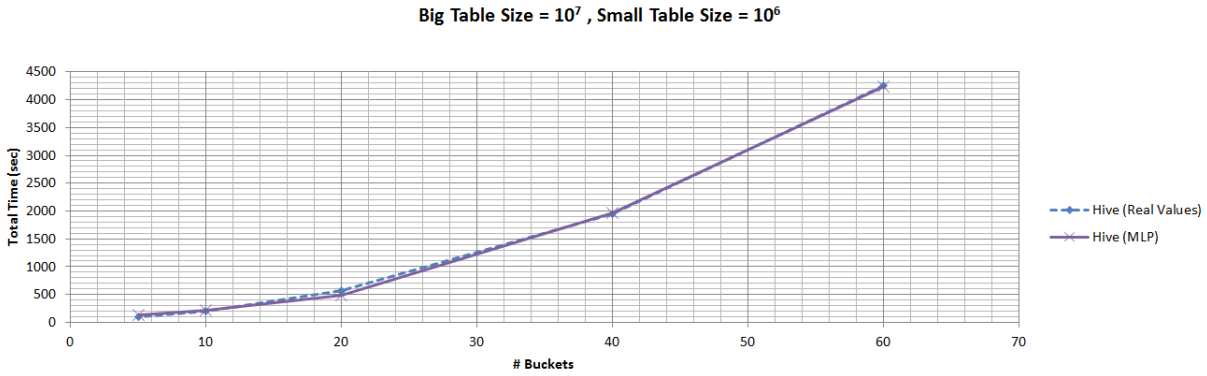


Σχήμα 4.15: Χρόνος Εκτέλεσης - Μέγεθος μικρού πίνακα (Map - side Bucket Join)

4.4.2.4 Μεταβαλλόμενος Αριθμός Κάδων

Παρατίθεται το μοντέλο (Σχ. 4.16) που φανερώνει την επίδραση του αριθμού των κάδων (buckets) που χρησιμοποιούνται από το bucket Join στον χρόνο εκτέλεσης. Η επίδοση χει-

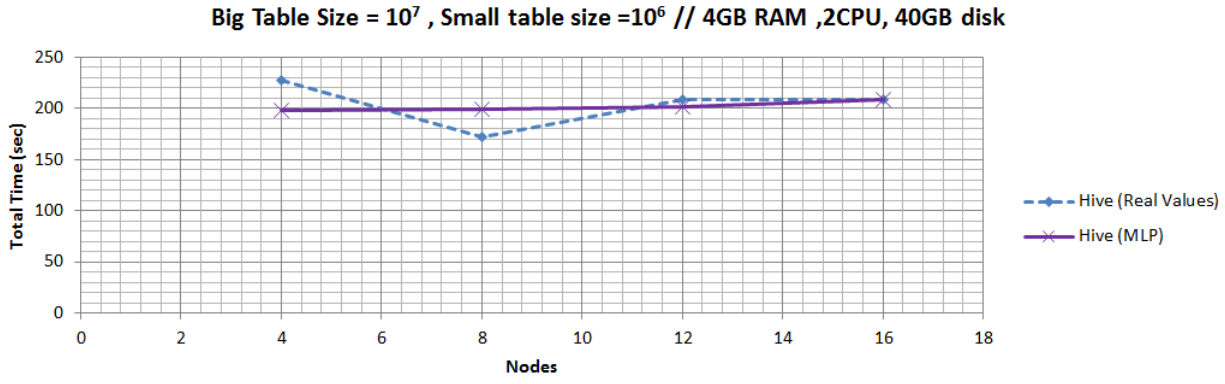
ροτερεύει με την αύξηση του αριθμού των κάδων. Αυτό συμβαίνει επειδή αυξάνονται κατά τάξεις μεγέθους οι διεργασίες map που εκτελούνται. Για παράδειγμα για αριθμών κάδων και στους δύο πίνακες 10, χρειάζονται 10 x 10 = 100 map διεργασίες. Από την άλλη πλευρά, το πλεονέκτημα χρησιμοποίησης πολλών κάδων, δίνει την δυνατότητα εκτέλεσης του map - side Join για μεγαλύτερα μεγέθη πίνακα.



Σχήμα 4.16: Χρόνος Εκτέλεσης - Αριθμός Κάδων (Map - side Bucket Join)

4.4.2.5 Μεταβαλλόμενος Αριθμός Κόμβων

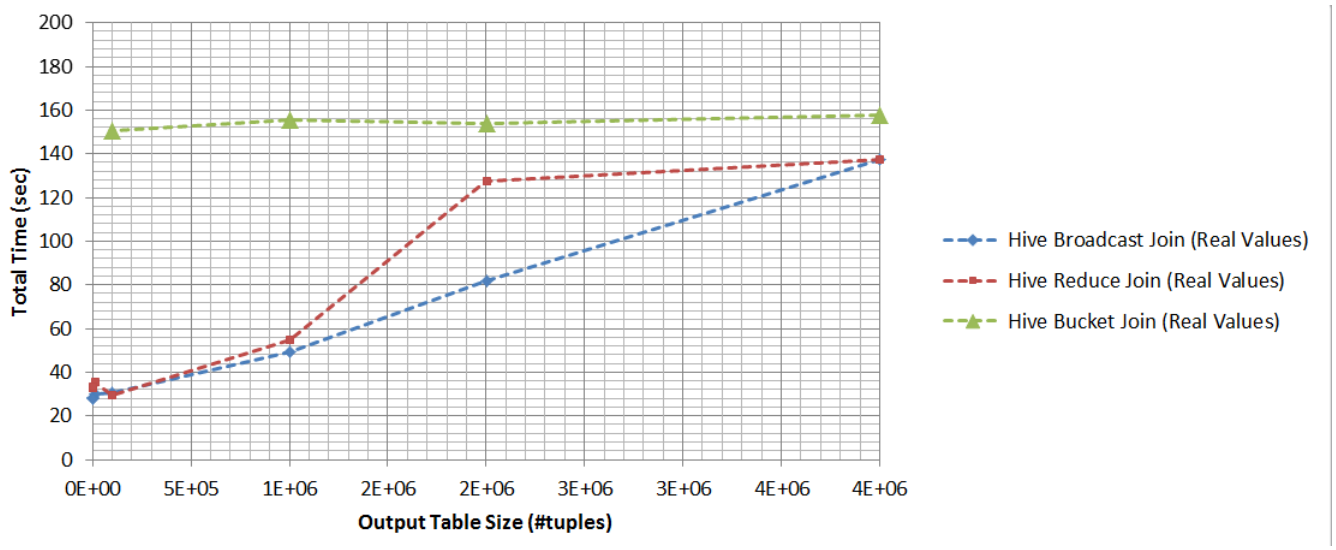
Για λόγους πληρότητας, αφού δεν πραγματοποιείται κάποια σύγκριση, παρατίθεται το μοντέλο για τον αριθμό των κόμβων, από το οποίο μπορεί να παρατηρήσει κανείς ότι το hive στον αλγόριθμο αυτό, σε αντίθεση με το broadcast join, δεν αλλάζει την επίδοσή του όσο ο αριθμός των κόμβων αυξάνεται. Από το γεγονός αυτό μπορεί να διαπιστωθεί ότι η επίδοση για το bucket Join είναι φραγμένη από παραμέτρους διαφορετικές του αριθμού των κόμβων. Ο τρόπος λειτουργίας του υποδεικνύει ότι το μέγεθος της μνήμης RAM αλλά και ο αριθμός των κάδων παίζουν πολύ σημαντικότερο ρόλο. Τέλος και εδώ ισχύει το συμπέρασμα για τον περιορισμένο αριθμό mapper που ίσχυε και στο broadcast Join.



Σχήμα 4.17: Χρόνος Εκτέλεσης - Αριθμός Κόμβων (Map - side Bucket Join)

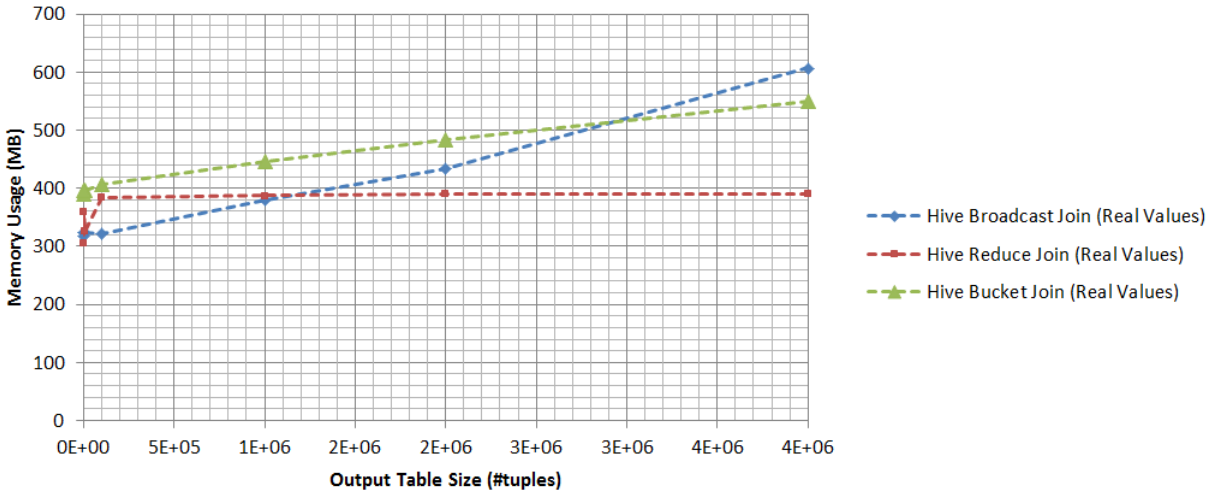
4.5 Σύγκριση των Αλγορίθμων Συνένωσης και Γενικά Σχόλια

Η σύγκριση που ακολουθεί (Σχ. 4.18) είναι για την εκτέλεση σε Hive που δεδομένων των αποτελεσμάτων, αποτελεί την βέλτιστη επιλογή σε σχέση με το Spark αλλά και με την postgresql (λόγω κλιμακοσιμότητας). Αναπαριστάται για κάθε αλγόριθμο ο συνολικός χρόνος εκτέλεσης για την Map - Reduce εργασία. Για τα μικρά σετ δεδομένων υπάρχει καλύτερη επίδοση του broadcast join σε σχέση με τον Reduce Side. Αυτό συμβαίνει επειδή ο μικρός πίνακας (του οποίου το μέγεθος εν προκειμένω ισούται με το μέγεθος του πίνακα εξόδου) μπαίνει στην cache και στέλνεται σε κάθε mapper. Το πλεονέκτημα φαίνεται να χάνεται στα μεγάλα σετ δεδομένων. Το overhead από τη διαδικασία του φορτώματος στην cache αυξάνεται προφανώς με την αύξηση του μεγέθους του μικρού πίνακα. Το bucket Join έχει την χειρότερη επίδοση. Το γεγονός αυτό οφείλεται στο κόστος της αποστολής σε κάθε διεργασία map του κομματιού του μικρού πίνακα που χρειάζεται. Ωστόσο αυτό που εύκολα διαπιστώνει κανείς, είναι ότι η κλίση είναι σαφέστατα μικρότερη στην περίπτωση του bucket Join, γεγονός που το καθιστά την καλύτερη επιλογή από ένα μέγεθος πίνακα και πάνω. Το μειονέκτημα του bucket Join για τα μικρότερα μεγέθη μπορεί να αντιμετωπιστεί με εφαρμογή του αλγορίθμου Sorted-Merge-Bucket Join, στον οποίο οι πίνακες ταξινομούνται κατά την τοποθέτησή τους στους κάδους με βάση το κλειδί της συνένωσης. Έτσι δεν χρειάζεται η αποστολή πολλών κάδων στον ίδιο mapper. Αξίζει να σημειωθεί ωστόσο ότι υπάρχει επιπλέον κόστος προ επεξεργασίας λόγω ταξινόμησης.



Σχήμα 4.18: Χρόνος Εκτέλεσης - Μέγεθος πίνακα εξόδου (συγκριτικό) σε Hive

Στο Σχ.4.19 γίνεται σύγκριση της χρησιμοποιούμενης μνήμης για διάφορα μεγέθη πίνακα εξόδου. Το Reduce Side Join φαίνεται να έχει σταθερή συμπεριφορά όσο αφορά τις απαιτήσεις σε μνήμη. Αντίθετα, οι αλγόριθμοι Map Side Join παρουσιάζουν εμφανή αυξητική τάση, η οποία οφείλεται στις συνεχώς αυξανόμενες απαιτήσεις μνήμης για αποθήκευση (caching) του μικρού πίνακα.



Σχήμα 4.19: Μνήμη - Μέγεθος πίνακα εξόδου για το Hive

Γενικά οι αλγόριθμοι Map Side είναι προτεινόμενοι στην περίπτωση όπου ο μικρός πίνακας της συνένωσης είναι αρκετά μικρότερος του μεγάλου και επίσης αρκετά μικρός γενικά, ώστε να χωράει στην μνήμη. Εάν οι δύο πίνακες έχουν παραπλήσια μεγέθη (όπως φαίνεται και από τα παραπάνω σχήματα) το κέρδος είναι μικρό, και μετά από κάποιο σημείο εξαλείφεται πλήρως.

Κεφάλαιο 5

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάζεται μια σύνοψη της παρούσας διπλωματικής και μια επισκόπηση των αποτελεσμάτων της. Ακόμα παρουσιάζονται ιδέες για μελλοντική βελτίωση και επέκταση του συστήματος όπως επίσης και σχετική έρευνα.

5.1 Σύνοψη και Γενικά Σχόλια

Η διπλωματική αυτή είχε ως στόχο την δημιουργία ενός συστήματος με το οποίο ένας χρήστης θα μπορούσε με βάση τα δεδομένα χαρακτηριστικά εισόδου της εργασία που θέλει να εκτελέσει, να επιλέξει την βέλτιστη από τις διαθέσιμες μηχανή εκτέλεσης για να το κάνει. Βασική και προφανής απαίτηση, είναι να πραγματοποιηθεί αυτό χωρίς την ανάγκη πειραμάτων αλλά χρησιμοποιώντας έτοιμα μοντέλα του συστήματος.

Δεδομένου του μεγάλου αριθμού παραμέτρων, η κάλυψη ολόκληρου του χώρου των δειγμάτων ήταν πρακτικά αδύνατη. Έτσι έγινε επιλογή των μοντέλων που δημιουργήθηκαν, ώστε να συνοψίζονται κατά το δυνατόν περισσότερο η ζητούμενη πληροφορία (ποια είναι η αποδοτικότερη μηχανή εκτέλεσης) στα παραχθέντα μοντέλα.

Αντιμετωπίστηκαν προβλήματα θορύβου, τα οποία αντιμετωπίστηκαν στο επίπεδο εξαγωγής του τελικού πορίσματος για την αποδοτικότερη μηχανή. Τα μοντέλα που παρουσιάστηκαν αποτελούν αυτά με το μεγαλύτερο ενδιαφέρον καθώς τα παραχθέντα μοντέλα στο σύνολό τους είναι αρκετά περισσότερα σε αριθμό. Έγινε προσπάθεια εξαγωγής μοντέλων και συμπερασμάτων για τα μεγαλύτερα δυνατά μεγέθη εισόδου. Οι δυνατότητες και οι πόροι του συμπλέγματος έθεσαν το άνω όριο.

5.2 Μελλοντικές Ιδέες και Σχετική Έρευνα

Το αμέσως επόμενο βήμα για την βελτίωση του συστήματος αυτού, είναι η περαιτέρω εκπαίδευση των μοντέλων μέσω της διεξαγωγής πειραμάτων για μεγαλύτερα μεγέθη παραμέτρων. Σημαντικά συμπεράσματα θα προκύψουν από ένα τέτοιο βήμα καθώς οι κατανεμημένες μηχανές αναζήτησης έχουν δώσει σημάδια ότι θα έχουν αρκετά καλή επίδοση για μεγάλα μεγέθη εισόδου.

Επίσης όπως παρουσιάστηκε και στην αρχιτεκτονική του συστήματος, θα πρέπει να δημιουργηθεί ένα σύστημα όπου θα λαμβάνει ως είσοδο τα μοντέλα, θα δημιουργεί και θα εκτελεί

ένα πλάνο εκτέλεσης με βέλτιστο τρόπο. Έτσι ο χρήστης δεν θα χρειάζεται καθόλου τεχνική γνώση επί των μηχανών εκτέλεσης, γεγονός πολύ σημαντικό, καθώς η διαδικασία χειροκίνητης εκτέλεσης απαιτεί την απαραίτητη εξοικείωση και εμπεριέχει σημαντικό κίνδυνο σφαλμάτων.

Τέλος μία ακόμη ιδέα για μελλοντική επέκταση, αποτελεί αυτοματοποίηση της διαδικασίας μοντελοποίησης που εκτελέστηκε χειροκίνητα στα πλαίσια αυτής της διπλωματικής, με στόχο την κάλυψη σημαντικά μεγαλύτερου μέρους του χώρου των δειγμάτων.

Βιβλιογραφία

[1] Determine yarn and mapreduce memory configuration settings. http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.9.1/bk_installing_manually_book/content/rpm-chap1-11.html.

[2] Hadoop wiki. https://wiki.apache.org/hadoop.

[3] Normal distribution — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Normaldistribution.

[4] Uniform distribution (continuous) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Uniform_distribution_(continuous).

[5] U. Dayal A. Simitsis, K. Wilkinson and M. Hsu. Hfms: Managing the lifecycle and complexity of hybrid analytic data flows. In ICDE. IEEE, 2013.

[6] Apache Software Foundation. Apache Hive Documentation, 1.1.0 edition.

[7] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. VLDB, 5:2014–2015, 2012.

[8] Christos Doulkeridis. Processing joins over big data in mapreduce. 2014.

[9] Christos Doulkeridis and Kjetil NØrvåg. A survey of large-scale analytical query processing in mapreduce. VLDB, 23:355–380, 2014.

[10] Nikolaos Papailiou Ioannis Giannakopoulos, Dimitrios Tsoumakos and Nectarios Koziris. Panic: Modeling application performance over virtualized resources. In IEEE International Conference on Cloud Engineering, 2015.

[11] Natacha Crooks Matthew P. Grosvenor Allen Clement Steven Hand Ionel Gog, Malte Schwarzkopf. Musketeer: all for one, one for all in data processing systems. In EuroSys '15 Proceedings of the Tenth European Conference on Computer Systems, 2015.

[12] Jian Pei Juawei Han, Micheline Kamber. DATA MINING : Concepts and Techniques. Morgan Kaufmann, 3rd edition, 2012.

[13] D. Tsoumakos C. Mantas N. Koziris K. Doka, N. Papailiou. Ires: Intelligent, multi-engine resource scheduler for big data analytics workflows. In ACM SIGMOD/PODS International Conference on Management of Data, 2015.

[14] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation and experience. Parallel Computing, 30:2004, 2003.

[15] Padraig O'Sullivan. Comparing postgresql 9.1 vs. mysql 5.6 using drupal 7.x. 2012.

[16] Srini Penchikala. Big data processing with apache spark – part 1: Introduction. 2015.

[17] The PostgreSQL Global Development Group. PostgreSQL Documentation, 9.4.2 edition.

[18] T.A. Reddy. Probability Concepts and ProbabilityDistributions, chapter 2. Springer, 2011.

[19] Vuk Ercegovac Jun Rao Eugene J. Shekita Yuanyuan Tian Spyros Blanas, Jignesh M. Patel. A comparison of join algorithms for log processing in mapreduce. SIGMOD, pages 975–986, 2010.

[20] Δημήτρη Σαρλής. Κατανεμημένοι Αλγόριθμοι Ερωτημάτων Ένωσης με Εφαρμογές στην Ανάλυση Δεδομένων Δικτυακής Κίνησης. Master's thesis, National Technical University of Athens, Electrical and Computer Engineering, 2014.