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

Εντοπισμός Ανωμαλιών σε Πόρους Υπολογιστικών Κόμβων

Διπλωματική Εργασία

Γεώργιος Παπαδημητρίου

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εργαστήριο Υπολογιστικών Συστημάτων
Αθήνα, Μάρτιος 2018

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Εντοπισμός Ανωμαλιών σε Πόρους Υπολογιστικών Κόμβων

Διπλωματική Εργασία

Γεώργιος Παπαδημητρίου

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Μαρτίου 2018.

.....	.....	.....
Νεκτάριος Κοζύρης	Νικόλαος Παπασπύρου	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Εργαστήριο Υπολογιστικών Συστημάτων

Αθήνα, Μάρτιος 2018

.....
Γεώργιος Παπαδημητρίου
Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα δημόσια και ιδιωτικά υπολογιστικά νέφη είναι πολύ δημοφιλής λύσεις απόκτησης πόρων και δημιουργίας συστάδων υπολογιστών για την εκτέλεση πολύπλοκων υπολογισμών και πολλές φορές πάνω σε μεγάλους όγκους δεδομένων. Ωστόσο η συντήρηση ή η αποκατάσταση ζημιών μπορεί να αποφέρει αρκετά μεγάλη σε διάρκεια διακοπή υπηρεσιών και να προκαλέσει οικονομικές ζημιές τόσο στον χρήστη του υπολογιστικού νέφους όσο και στον πάροχο. Στόχος της παρούσας διπλωματικής είναι ο εντοπισμός ανωμαλιών σε υπολογιστικούς κόμβους, παρακολουθώντας τη χρησιμοποίηση των πόρων τους. Για το σκοπό αυτό δημιουργούμε ένα σύνολο δεδομένων που παρουσιάζει περιοδικότητα, χρησιμοποιώντας τη σουίτα αξιολόγησης συστημάτων επεξεργασίας δεδομένων μεγάλου όγκου HiBench. Επίσης αξιολογούμε τον εντοπισμό ανωμαλιών σε δεδομένα από το σύνολο δεδομένων Google Cluster Data. Τέλος υλοποιούμε σε Apache Spark μια έκδοση εντοπισμού ανωμαλιών βασισμένη στον αλγόριθμο Robust PCA και αξιολογούμε την δυνατότητα κλιμάκωσής του.
Από τα ευρήματά μας η εφαρμογή μιας μεθόδου, όπως του Robust PCA, για τον μη επιβλεπόμενο εντοπισμό ανωμαλιών σε δεδομένα χρονοσειρών, απαιτεί την ύπαρξη περιοδικότητας και στις περιπτώσεις που λείπει δημιουργούνται πολλές λανθασμένες ενδείξεις. Από την άλλη, η υλοποίηση σε Apache Spark παρέχει σχεδόν γραμμική επιτάχυνση με την αύξηση των εργατών, στον χρόνο υπολογισμού των πιθανών σημείων ανωμαλιών.

Λέξεις Κλειδιά

Εντοπισμός Ανωμαλιών, Μη Επιβλεπόμενος Εντοπισμός Ανωμαλιών, Robust Principal Component Analysis, Εισαγωγή Ανωμαλιών, Υπολογιστικοί Κόμβοι, Πόροι Συστήματος, HiBench, Google Cluster Data, Apache Spark, Scala, Breeze

Abstract

Public and private cloud infrastructures have become a very popular way of acquiring resources on demand and deploying computing clusters that perform complex computations, and most of the time on large amount of data. However, maintenance or recovering from failures can be the reason of significant service downtime and can impact economically both the cloud platform user and the service provider. The goal of this work is the identification of anomalous events on compute nodes, by monitoring their resource usage. In order to study this, we create a dataset that presents seasonality in its data, using the big data benchmark suite "HiBench", where we inject resource anomalies at certain points. On this dataset we apply an unsupervised anomaly detection technique called Robust Principal Component Analysis, in order to identify the anomalies in the data set. Additionally, we reconstruct the resource usage data from the "Google Cluster Data" dataset on a per node basis, and we evaluate whether we can apply a similar technique. Finally, we implement a version of the algorithm in Apache Spark and evaluate its ability to scale by adding additional workers in the Spark cluster.
From our tests, it turns out that the application of Robust PCA, as an anomaly detection technique, relies heavily on the existence of seasonality and its correct identification for the analysis. Moreover, this technique cannot be applied to the Google Cluster Data, due to lack of seasonality on the majority of the compute nodes and the extremely dynamic workload of the system. To conclude, the Apache Spark implementation seems to be scaling almost linearly in our tests, and this is because of the coarse grained approach we took in the solution of the problem.

Keywords

Anomaly Detection, Unsupervised Anomaly Detection, Robust Principal Component Analysis, Anomaly Injection, Compute Nodes, System Resources, HiBench, Google Cluster Data, Apache Spark, Scala, Breeze

Ευχαριστίες

Με τη διπλωματική αυτή εργασία σηματοδοτείται το τέλος της φοίτησής μου στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσοβίου Πολυτεχνείου.
Αρχικά, θα ήθελα να πω ένα μεγάλο ευχαριστώ στην οικογένειά μου που με στήριξε σε όλη τη διάρκεια των σπουδών μου, αλλά και στους στενούς μου φίλους που ήταν δίπλα μου και με βοήθησαν στην επίτευξη των στόχων μου.
Έπειτα, θα ήθελα να ευχαριστήσω όλους τους καθηγητές των τομέων Λογισμικού και Υπολογιστικών Συστημάτων, καθώς η αγάπη τους για το αντικείμενό τους και η υψηλή ποιότητα των μαθημάτων ήταν βασικός παράγοντας για την εξέλιξή μου ως φοιτητής και ως μηχανικός, αλλά και για το όλο και αυξανόμενο ενδιαφέρον μου στα υπολογιστικά συστήματα.
Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον Καθηγητή Ε.Μ.Π Νεκτάριο Κοζύρη για την ευκαιρία που μου έδωσε να εκπονήσω τη διπλωματική μου εργασία πάνω σε ένα πεδίο που αγαπώ και επιθυμώ να συνεχίσω τις μετέπειτα σπουδές μου.
Τέλος, δε γίνεται να ξεχάσω τον ερευνητή Ιωάννη Κωνσταντίνου για την καθοδήγηση και τη βοήθεια που μου παρείχε κατά τη διάρκεια αυτής της εργασίας, αλλά και τον υποψήφιο διδάκτορα Ιωάννη Γιαννακόπουλο για τη βοήθεια και τις συμβουλές του σε τεχνικά ζητήματα που αντιμετώπισα.

Γεώργιος Παπαδημητρίου

Αθήνα, Μάρτιος 2018

Περιεχόμενα

List of Algorithms

Κατάλογος σχημάτων

Κατάλογος πινάκων

Λίστα των Scripts

Κεφάλαιο 1

Εισαγωγή

Τα τελευταία χρόνια η άνοδος των υπολογιστικών νεφών και υπηρεσιών παροχής υποδομής (IaaS) και πλατφόρμας (PaaS), έχουν κάνει αισθητή την παρουσία τους. Όλο και περισσότερες είναι οι εταιρείες που επιλέγουν να μεταφέρουν τους εξυπηρετητές και τις εσωτερικές υπηρεσίες τους, από το ιδιωτικό τους δίκτυο σε υπηρεσίες όπως το Amazon AWS και το Microsoft Azure. Startups τεχνολογίας επιταχύνουν τη δημιουργία του προϊόντος τους, εκμεταλλευόμενες τις υπηρεσίες παροχής πλατφόρμας, αποκτώντας υποδομή άμεσα και με μικρότερο κόστος. Διαδικτυακές εφαρμογές με εκατομμύρια χρήστες επιλέγουν να νοικιάσουν υποδομή και να εξυπηρετούν τους χρήστες τους από διαφορετικά κέντρα, για να τους παρέχουν καλύτερη εμπειρία χρήσης και για να μειώσουν το λειτουργικό τους κόστος (πχ. Netflix, Slack). Επιστήμονες και ερευνητές που δεν έχουν τοπική υποδομή για την εκτέλεση των προσομοιώσεών τους, στρέφονται σε υπολογιστικά νέφη (διαπανεπιστημιακά, δημόσια και ιδιωτικά) που τους παρέχουν μονάδες επεξεργασίας ειδικού σκοπού και επιταχύνουν το έργο τους.

1.1 Κίνητρο

Ωστόσο αυτές οι υποδομές, αν και παρέχουν μεγάλη αξιοπιστία υλοποιώντας διάφορες δικλείδες ασφαλείας, αντιμετωπίζουν προβλήματα βλαβών που μπορεί να διαταράξουν την παροχή υπηρεσιών και κατά συνέπεια την κανονική λειτουργία των εργασιών των χρηστών τους. Αυτά τα προβλήματα συνήθως οφείλονται κυρίως σε αμέλεια του ανθρώπινου παράγοντα, όπως λανθασμένη εγκατάσταση, λάθος συντήρηση, λάθος διαχείριση ή λάθος ρύθμιση. Άλλες φορές όμως τα προβλήματα προέρχονται από ανεπάντεχες αστοχίες υλικού, όπως καρτών δικτύου, τροφοδοτικών, σκληρών δίσκων. Ενώ, άλλες φορές προέρχονται από την ύπαρξη κακόβουλου λογισμικού.
Τα συμπτώματα αυτά όμως γίνονται πιο έντονα σε περιπτώσεις ιδιωτικών υπολογιστικών εγκαταστάσεων μερικών δεκάδων κόμβων. Τέτοιες εγκαταστάσεις τείνουν να αμελούν δικλείδες ασφαλείας (όπως πλεονασμό σε δίσκους) για να μειώσουν το κόστος.
Έτσι, ο εντοπισμός ανωμαλιών σε αυτά τα συστήματα φαίνεται να είναι πολύ σημαντικός, καθώς μπορεί να υποδείξει σε τι οφείλεται η δυσλειτουργία ή η αστοχία του συστήματος, αλλά και σε ορισμένες περιπτώσεις να δώσει πληροφορία για μια βλάβη που ακόμα δεν έχει

πάρει την πλήρη έκτασή της.
Ωστόσο τέτοιες εγκαταστάσεις συνηθίζεται να αναλαμβάνουν την εκτέλεση ποικίλων εργασιών οι οποίες συνέχεια εξελίσσονται και αλλάζουν. Είναι εφικτό, λοιπόν, να παρατηρήσουμε αποδοτικά ανωμαλίες σε ένα τέτοιο περιβάλλον μόνο παρακολουθώντας την χρήση πόρων του συστήματος;

1.2 Στόχος της διπλωματικής εργασίας

Έτσι, στόχος αυτή της διπλωματικής εργασίας είναι να εξετάσουμε τον εντοπισμό ανωμαλιών σε ένα περιβάλλον συστάδων υπολογιστών, με τρόπο που να μη χρειάζεται αυστηρή μοντελοποίηση και επίβλεψη της συμπεριφοράς των μηχανημάτων και να μην επιφέρει πολλές εσφαλμένες ειδοποιήσεις. Ενώ επιπλέον θέτουμε σαν προϋπόθεση να μη γνωρίζουμε τη φύση και την ακριβή συμπεριφορά των εργασιών που εκτελούνται.

1.3 Συνεισφορά της διπλωματικής εργασίας

Για το σκοπό αυτό δημιουργούμε ένα σύνολο δεδομένων, από παρακολούθηση πόρων μιας ιδιωτικής συστάδας υπολογιστών, το οποίο συλλέγουμε για περίπου 2 εβδομάδες. Ανακτούμε και επεξεργαζόμαστε τα δεδομένα που παρέχονται για τους πόρους συστήματος στο σύνολο δεδομένων Google Cluster Data και τα ανάγουμε σε κάθε εικονικό μηχάνημα. Δοκιμάζουμε μια τεχνική εύρεσης ανωμαλιών με βάσει τον αλγόριθμο Robust Principal Component Analysis, την οποία υλοποιούμε και στο Apache Spark. Τέλος παρουσιάζουμε μετρήσεις και δοκιμές γύρω από αυτές τις υλοποιήσεις.

1.4 Οργάνωση της διπλωματικής εργασίας

Η οργάνωση που ακολουθεί η διπλωματική εργασία είναι η ακόλουθη:

Κεφάλαιο 2: Παρουσιάζουμε το απαραίτητο θεωρητικό και τεχνολογικό υπόβαθρο, για να μπορέσει ο αναγνώστης να κατανοήσει βασικές αρχές και μηχανισμούς, που σχετίζονται με αυτή τη διπλωματική εργασία.

Κεφάλαιο 3: Παρουσιάζουμε τον αλγόριθμο που θα χρησιμοποιήσουμε για την εύρεση των ανωμαλιών.

Κεφάλαιο 4: Παρουσιάζουμε τα σύνολα δεδομένα που χρησιμοποιήθηκαν για την αποτίμηση του αλγορίθμου και την απόδοση του τελικού συστήματος.

Κεφάλαιο 5: Παρουσιάζουμε την υλοποίηση της αποδόμησης πινάκων που χρησιμοποιείται για τον εντοπισμό των ανωμαλιών στο Apache SPARK.

Κεφάλαιο 6: Παρουσιάζουμε αποτελέσματα προσομοιώσεων για την απόδοση του συστήματος ως προς την εύρεση ανωμαλιών.

Κεφάλαιο 7: Κάνουμε μια σύνοψη των αποτελεσμάτων και προτείνουμε μελλοντικές εργασίες.

Κεφάλαιο 2

Θεωρητικό και Τεχνολογικό Υπόβαθρο

2.1 Αλγόριθμοι Εύρεσης Ανωμαλιών

Μέσα στα χρόνια έχουν χρησιμοποιηθεί διάφορες τεχνικές για την εύρεση ανωμαλιών σε πολλές εφαρμογές. Τις τεχνικές αυτές μπορούμε να τις εντάξουμε σε τρεις μεγάλες κατηγορίες. Τη μη επιβλεπόμενη εύρεση ανωμαλιών (Unsupervised Anomaly Detection), την επιβλεπόμενη εύρεση ανωμαλιών (Supervised Anomaly Detection) και την ημι-επιβλεπόμενη εύρεση ανωμαλιών (Semi-supervised Anomaly Detection).

- Τεχνικές Unsupervised Anomaly Detection εφαρμόζονται σε σύνολα δεδομένων για τα οποία δεν έχουμε καμία ένδειξη. Για την εύρεση των ανωμαλιών γίνεται η υπόθεση ότι το μεγαλύτερο μέρος των δεδομένων είναι φυσιολογικά και θεωρούν ανωμαλίες αυτά που δε ταιριάζουν με τον κύριο όγκο των δεδομένων.
- Οι τεχνικές Supervised Anomaly Detection απαιτούν δεδομένα, τα οποία έχουν μαρκαριστεί με ετικέτες και χωριστεί σε φυσιολογικά και αφύσικα. Αυτές οι τεχνικές συνήθως περιλαμβάνουν την εκπαίδευση ενός ταξινομητή (classifier) με τα φυσιολογικά και τα αφύσικα δεδομένα και την ανάπτυξη ενός μοντέλου που στην συνέχεια μπορεί να χαρακτηρίζει νέα δεδομένα σαν ανώμαλα ή όχι.
- Οι τεχνικές Semi-supervised Anomaly Detection απαιτούν ένα σύνολο δεδομένων που περιέχει μόνο φυσιολογικά σημεία. Αυτές οι τεχνικές περιλαμβάνουν την κατασκευή μοντέλου από τα φυσιολογικά δεδομένα και δοκιμάζουν την πιθανότητα νέα δεδομένα να είναι ανώμαλα ή όχι, βάσει της εξόδου του μοντέλου.

Στην παρούσα διπλωματική, ωστόσο, ασχολούμαστε με τεχνικές μη επιβλεπόμενης εύρεσης ανωμαλιών πάνω σε δεδομένα χρονοσειρών. Στη βιβλιογραφία για την εύρεση ανωμαλιών χωρίς επίβλεψη, ανωμαλία θεωρείται κάποιο σημείο του συνόλων δεδομένων, το οποίο αποκλίνει από τη νόρμα.
Οι διάφορες τεχνικές και αλγόριθμοι μη επιβλεπόμενου εντοπισμού ανωμαλιών, μπορούν να ενταχθούν σε πέντε βασικές ομάδες.

- Τεχνικές βασισμένες στον κοντινότερο γείτονα (Nearest-Neighbor)
- Τεχνικές βασισμένες στην ομαδοποίηση (Clustering)
- Τεχνικές βασισμένες στη στατιστική ανάλυση (Statistical Analysis)
- Τεχνικές βασισμένες στην εύρεση υποχώρων (Subspace)
- Τεχνικές βασισμένες στην κατάταξη (Classification) των δεδομένων

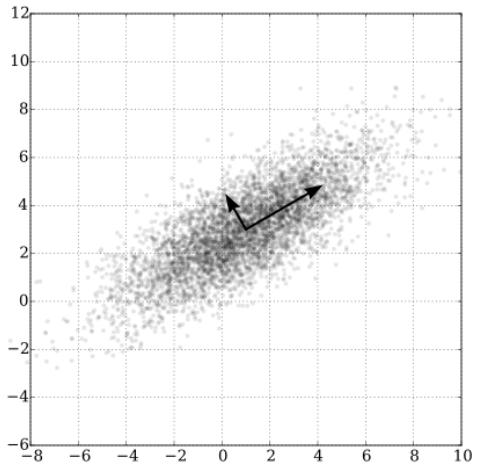
Από τις παραπάνω ομάδες επιλέγουμε να αναφερθούμε εκτενέστερα σε τεχνικές εύρεσης υπόχωρων, και συγκεκριμένα στους αλγορίθμους Principal Component Analysis και Robust Principal Component Analysis, καθώς θα μας φανούν χρήσιμοι στην πορεία της διπλωματικής εργασίας.

2.1.1 Principal Component Analysis (PCA)

Η Ανάλυση σε Κύριες Συνιστώσες (Principal Component Analysis) ανακαλύφτηκε από τον Karl Pearson το 1901,σαν αναλογία του θεωρήματος αξόνων της μηχανικής. Τη δεκαετία του 1930 αναπτύχθηκε ανεξάρτητα από τον Harol Hotelling, ο οποίος έδωσε και όνομα στην τεχνική αυτή.
Η ανάλυση σε κύριες συνιστώσες είναι μια στατιστική διαδικασία που μετατρέπει ένα σύνολο (πιθανά εξαρτημένων) μεταβλητών σε ένα σύνολο τιμών γραμμικά ανεξάρτητων μεταβλητών, που καλούνται Κύριες Συνιστώσες (Principal Components).
Η τεχνική αυτή χρησιμοποιείται κυρίως σαν εργαλείο για την διερευνητική ανάλυση δεδομένων και για τη δημιουργία μοντέλων πρόβλεψης. Ο υπολογισμός των κύριων συνιστωσών μπορεί να διενεργηθεί είτε με την αποσύνθεση των ιδιοτιμών του πίνακα συσχέτισης των δεδομένων, είτε με την αποσύνθεση μοναδικής τιμής του πίνακα δεδομένων, συνήθως μετά την κανονικοποίηση των δεδομένων.

Βασικά χαρακτηριστικά αυτής της τεχνικής είναι:

- Το πλήθος των κύριων συνιστωσών μπορεί να είναι ίσο ή μικρότερο του πλήθους των αρχικών μεταβλητών.
- Η πρώτη κύρια συνιστώσα έχει τη μεγαλύτερη δυνατή διακύμανση.
- Όλες οι κύριες συνιστώσες είναι διατεταγμένες σε φθίνουσα σειρά, όσον αφορά τη διακύμανση που περιγράφουν.
- Κάθε κύρια συνιστώσα είναι κάθετη με όλες τις προηγούμενες.



Σχήμα 2.1: PCA μιας πολυμεταβλητής γκαουσιανής κατανομής

Ο PCA [22] μπορεί να θεωρηθεί ως ένας ορθογώνιος γραμμικός μετασχηματισμός που μετασχηματίζει τα δεδομένα σε νέο σύστημα συντεταγμένων, όπου η μεγαλύτερη διακύμανση των προβολών των δεδομένων βρίσκεται στη πρώτη συντεταγμένη του συστήματος και ακολουθούν οι υπόλοιπες με φθίνουσα σειρά.

2.1.2 Robust Principal Component Analysis (RPCA)

Ο PCA, αν και το πιο διαδεδομένο στατιστικό εργαλείο ανάλυσης δεδομένων και μείωσης των διαστάσεών τους, έχει ένα μεγάλο μειονέκτημα, είναι πολύ επιρρεπής σε αλλοιωμένες τιμές δεδομένων που μπορεί να θέσουν την εγκυρότητά του σε αμφιβολία.
Με αυτή την αφορμή έχουν γίνει αρκετές προσπάθειες και έχουν προταθεί αρκετές τεχνικές απο διάφορους ερευνητές για να διορθωθεί αυτό το πρόβλημα. Το 2011 ο Emanuel Candes και άλλοι, έκαναν μια δημοσίευση με τίτλο "Robust Principal Component Analysis?" [7] προσπαθώντας να αντιμετωπίσουν το πρόβλημα αυτό του PCA.
Σε αυτή τη δημοσίευση προτείνουν την εξής ιδέα:

- Έστω ότι έχουμε έναν πίνακα πολυμεταβλητών δεδομένων $M \in \mathbb{R}^{n_1 \times n_2}$, κάνοντας πολύ χαλαρές υποθέσεις, μπορούμε να αποδομήσουμε αυτόν τον πίνακα σε δύο πίνακες $L_0 \in \mathbb{R}^{n_1 \times n_2}$ και $S_0 \in \mathbb{R}^{n_1 \times n_2}$, όπου ο πρώτος αποτελεί τη συνιστώσα χαμηλής τάξης (Low Rank Component) του M και ο δεύτερος αποτελεί την αραιά συνιστώσα (Sparse Component) του M.

Δείχνουν ότι αυτή η αποσύνθεση μπορεί υπολογιστεί αποδοτικά μέσω κυρτής βελτιστοποίησης (Tructable Convex Optimization) και προτείνουν τον αλγόριθμο Principal Component Pursuit (PCP) που επιλύει προσεγγιστικά το πρόβλημα:

$$\begin{array}{ll} \text{ελαχιστοποίησε} & \|L\|_* + \lambda\|S\|_1 \\ \text{με στόχο} & L + S = M \end{array}$$

Όπου ο όρος $\|L\|_*$ αντιστοιχεί στην nuclear norm του L και ο όρος $\|S\|_1$ στην πρώτη νόρμα του S. Το λ είναι μια μεταβλητή που ορίζεται ίση με $1/\sqrt{n}$, με το $n = max(n_1, n_2)$.

Η υλοποιήση του Principal Component Pursuit (PCP) μπορεί να γίνει με διάφορες μεθόδους, όπως:

- Alternating Lagrange Multiplier (ALM) [7]
- Accelerated Proximal Gradient (APG) [13]
- Alternating Direction Method of Multipliers (ADMM) [30]

Ένα παράδειγμα εκτέλεσης της αποδόμησης πινάκων δίνεται παρακάτω:

M

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.793	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	0.856	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.1	0.2	0.3	0.124	0.5	0.6	0.7	0.8	0.9
0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

L

0.1128	0.1949	0.2548	0.2956	0.4247	0.5098	0.5946	0.6788	0.7647
0.1136	0.1962	0.2564	0.2975	0.4275	0.5132	0.5985	0.6832	0.7696
0.114	0.197	0.2574	0.2987	0.4291	0.5151	0.6008	0.6858	0.7726
0.109	0.1883	0.2461	0.2855	0.4102	0.4925	0.5744	0.6557	0.7386
0.1128	0.1949	0.2548	0.2956	0.4247	0.5098	0.5946	0.6788	0.7647

S

0	0	0	0	0	0	0	0	0
0.5296	0	0	0	0	0	0	0	0
0	0.5094	0	0	0	0	0	0	0
0	0	0	-0.01	0	0	0	0	0.0199
0	0	0	0	0	0	0	0	0

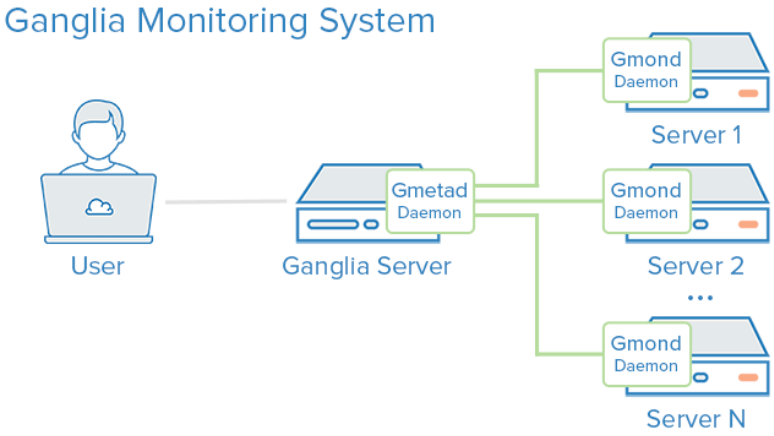
Σχήμα 2.2: Παράδειγμα αποδόμησης πινάκων με Robust PCA

2.2 Συστήματα Παρακολούθησης

2.2.1 Ganglia

Το Ganglia είναι ένα λογισμικό ανοιχτού κώδικα, διαθέτει BSD license και η ανάπτυξή του ξεκίνησε στο University of California, Berkeley. Αποτελεί ένα κλιμακώσιμο κατανεμημένο σύστημα παρακολούθησης για υπολογιστικά συστήματα υψηλής απόδοσης, όπως υπολογιστικά συστήματα πλέγματος (Grid). Εκμεταλλεύεται ευρέως διαδεδομένες τεχνολογίες, όπως XML (Extensible Markup Language) για την αναπαράσταση των δεδομένων, XDR (XML-Data Reduced) για τη συμπαγή φορητότητα των δεδομένων, και RRDtool για

την αποθήκευση και απεικόνιση των δεδομένων.
Χρησιμοποιώντας προσεκτικά επιλεγμένες δομές δεδομένων και αλγορίθμους, το Ganglia πετυχαίνει να έχει πολύ μικρή επιβάρυνση ανά κόμβο και πολύ μεγάλο συγχρονισμό. Είναι διαθέσιμο για πολλές αρχιτεκτονικές (i386, x86_64, ia64, sparc, alpha, powerpc, m68k, mips, arm, hppa, s390) και χρησιμοποιείται για την παρακολούθηση χιλιάδων συστημάτων παγκοσμίως.



Σχήμα 2.3: Μια τυπική αρχιτεκτονική του Ganglia

Το Ganglia αποτελείται από πέντε εργαλεία:

- Ganglia Meta Daemon (gmetad), το οποίο συλλέγει περιοδικά τις μετρήσεις από τους κόμβους.
- Ganglia Monitoring Daemon (gmond), το οποίο έχει τη δυνατότητα να παρακολουθεί διάφορα υποσυστήματα ενός υπολογιστικού κόμβου, όπως τον επεξεργαστή, την μνήμη, τον δίσκο και το δίκτυο, και να τα αναφέρει κεντρικά στο Ganglia Meta Daemon.
- Ganglia Metrics (gmetric), το οποίο είναι μια εφαρμογή γραμμής εντολών που χρησιμεύει για την εισαγωγή στη βάση του Ganglia παραμετροποιημένων μετρήσεων από τον χρήστη, σχετικά με τους κόμβους που παρακολουθούνται.
- Ganglia Statistics (gstat), το οποίο είναι μια εφαρμογή γραμμής εντολών που χρησιμεύει στην αναζήτηση και ανάσυρση μετρήσεων από τη βάση δεδομένων του Ganglia.
- Ganglia PHP Web Frontend, το οποίο παρέχει τη δυνατότητα απεικόνισης των δεδομένων σε πραγματικό χρόνο με ουσιαστικό τρόπο για τους διαχειριστές και τους χρήστες.

2.2.2 Άλλες πηγές μετρήσεων

Πέρα από το Ganglia, υπάρχουν κι άλλες πηγές μετρήσεων που μπορεί να δίνουν σημαντικές πληροφορίες για την κατάσταση μιας συστάδας υπολογιστικών κόμβων (Cluster). Τέτοιες πηγές μπορεί να είναι ο Cluster Hypervisor του συστήματός, αλλά και ο Cluster Job Scheduler.

2.3 Μηχανές και Εργαλεία Εκτέλεσης

2.3.1 Apache Hadoop

Το Apache Hadoop [2] είναι ένα framework που επιτρέπει την κατανεμημένη επεξεργασία μεγάλων ομάδων αρχείων σε συστάδες υπολογιστών χρησιμοποιώντας απλά προγραμματιστικά μοντέλα. Είναι σχεδιασμένο να κλιμακώνει από έναν server σε χιλιάδες, όπου ο καθένας τους προσφέρει αποθηκευτικό χώρο και πόρους για τοπικούς υπολογισμούς. Το Hadoop αντί να στηρίζεται στο υλικό για να παρέχει υψηλή διαθεσιμότητα, έχει σχεδιαστεί να εντοπίζει και να χειρίζεται της αστοχίες στο επίπεδο της εφαρμογής. Έτσι μπορεί να παρέχει υψηλή διαθεσιμότητα ενώ τρέχει πάνω σε μια συστάδα υπολογιστών (cluster of computers), οι οποίοι μπορεί να είναι επιρρεπείς σε αστοχίες υλικού.

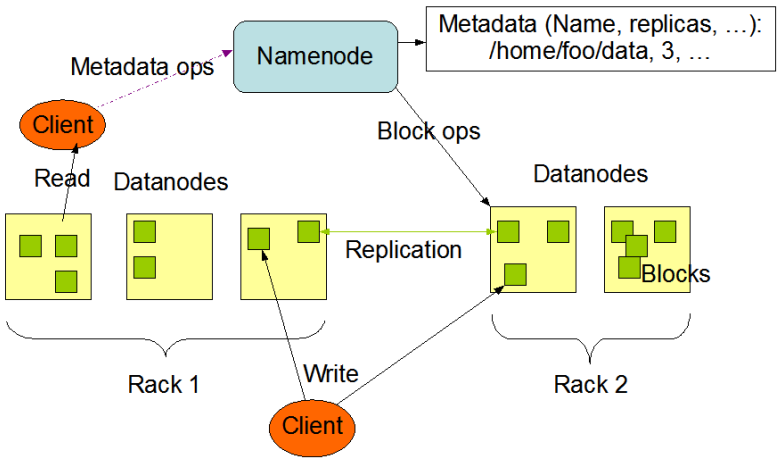
Το Apache Hadoop αποτελείται από τις παρακάτω μονάδες:

Hadoop Common

Το Hadoop Common περιλαμβάνει τα βοηθητικά προγράμματα που υποστηρίζουν τις άλλες μονάδες του Hadoop.

Hadoop Distributed File System

Το Hadoop Distributed File System (HDFS) είναι ένα κατανεμημένο σύστημα αρχείων, σχεδιασμένο να λειτουργεί πάνω σε υλικό που βρίσκουμε εύκολα στην αγορά. Έχει πολλές ομοιότητες με άλλα υπάρχοντα κατανεμημένα συστήματα αρχείων, αλλά οι διαφορές του είναι αρκετά μεγάλες. Το HDFS είναι πολύ ανεκτικό σε σφάλματα και έχει σχεδιαστεί για να τρέχει σε υλικό χαμηλού κόστους. Μπορεί να αποθηκεύσει πολύ μεγάλα αρχεία σε ένα πλήθος κόμβων εντός του cluster του, πράγμα που το κάνει πολύ καλή επιλογή για την αποθήκευση πολύ μεγάλων data sets, ενώ παράλληλα παρέχει υψηλή απόδοση στην πρόσβαση των δεδομένων.
Το HDFS αρχικά αναπτύχθηκε για να χρησιμοποιηθεί ως υποδομή για τη μηχανή αναζήτησης Apache Nutch, αλλά πλέον έχει γίνει κομμάτι του Hadoop framework.

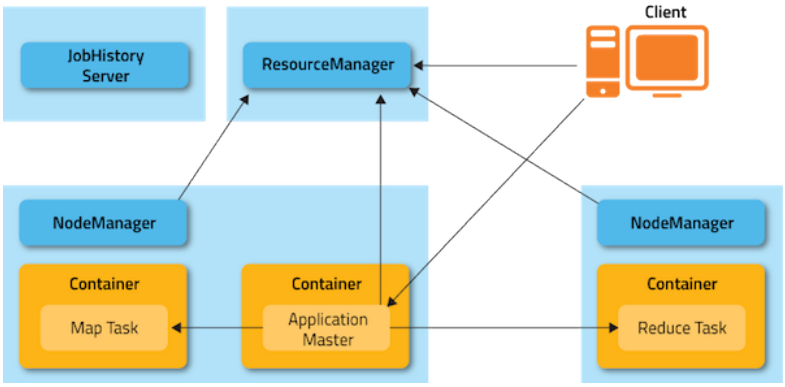


Σχήμα 2.4: Η αρχιτεκτονική του HDFS

Στηρίζει ένα παραδοσιακό ιεραρχικό μοντέλο οργάνωσης των αρχείων, που μοιάζει αρκετά με πολύ διαδεδομένα υπάρχοντα συστήματα αρχείων, όπως εκείνο του Unix, τουλάχιστον στα μάτια του χρήστη. Ωστόσο εσωτερικά το HDFS διατηρεί κάθε αρχείο σαν μια σειρά από blocks ίδιου μεγέθους (εκτός ίσως από το τελευταίο), τα οποία ανατίθενται και αναπαράγονται για ανοχή σε σφάλματα σε παραμετροποιήσιμο αριθμό κόμβων του HDFS cluster. Το HDFS διαθέτει αρχιτεκτονική κύριου/δευτερεύοντος (master/slave). Ένα HDFS cluster αποτελείται από έναν NameNode και τα DataNodes. Ο NameNode είναι ο κύριος server που διαχειρίζεται το χώρο ονομάτων του συστήματος αρχείων, αλλά και ρυθμίζει την πρόσβαση των πελατών στα αρχεία. Στη δικαιοδοσία του NameNode είναι η εκτέλεση λειτουργιών όπως άνοιγμα/κλείσιμο αρχείων και μετονομασία αρχείων και καταλόγων. Επιπλέον ο NameNode ορίζει με ποιον τρόπο θα μοιραστούν τα αρχεία στο cluster. Με τη σειρά τους οι DataNodes, είναι εκείνοι που διαχειρίζονται τον αποθηκευτικό χώρο πάνω στον κόμβο του cluster που τρέχουν. Είναι υπεύθυνοι για να εξυπηρετούν αιτήσεις εγγραφής και ανάγνωσης από τους πελάτες, και να εκτελούν εντολές από τον NameNode σχετικά με την δημιουργία, διαγραφή και αναπαραγωγή στα block των αρχείων.

Hadoop YARN

Το Hadoop YARN βρίσκεται στην καρδιά του Hadoop και αποτελεί βασικό του στοιχείο, που επιτρέπει σε άλλες μηχανές επεξεργασίας δεδομένων να έχουν πρόσβαση στα δεδομένα που βρίσκονται σε ένα HDFS cluster.
Το YARN μπορεί να χαρακτηριστεί σαν ένα framework χρονοδρομολόγησης εργασιών (job scheduler) και διαχείρισης πόρων μιας συστάδας υπολογιστών (cluster resource manager).



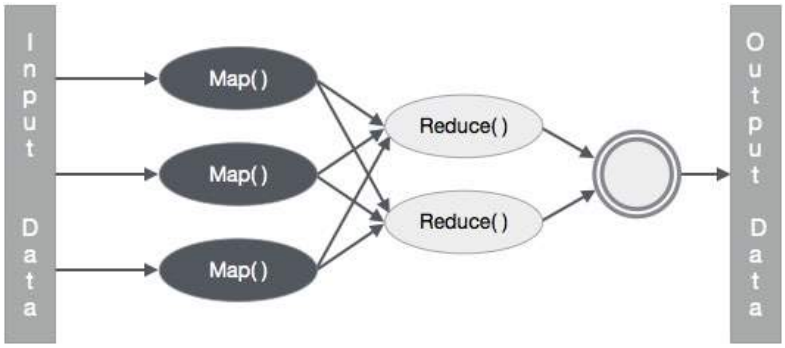
Σχήμα 2.5: Αρχιτεκτονική του YARN

Βασικά μέρη του Hadoop YARN είναι:

- **ResourceManager**: Ο ResourceManager είναι η κυρίαρχη αρχή που διαχειρίζεται τους πόρους για όλες τις εφαρμογές στο cluster. Ο ResourceManager διαθέτει έναν δρομολογητή ο οποίος είναι υπεύθυνος για την ανάθεση πόρων σε κάθε εφαρμογή που τρέχει στο cluster, λαμβάνοντας υπόψιν του περιορισμούς του cluster και όρια που έχει θέσει ο χρήστης.
- **NodeManager**: Ένας NodeManager τρέχει σε κάθε κόμβο του cluster και είναι υπεύθυνος για την εκκίνηση των containers των εφαρμογών, την παρακολούθηση των πόρων που χρησιμοποιούν, και τα οποία αναφέρει στον ResourceManager.
- **ApplicationMaster**: Ο ApplicationMaster είναι μια οντότητα που διαπραγματεύεται τους πόρους μιας εφαρμογής, με τον ResourceManager και συνεργάζεται με τους NodeManagers για να εκτελεστούν οι επιμέρους διεργασίες της εφαρμογής. Κάθε εφαρμογή διαθέτει τον δικό της ApplicationMaster και από την οπτική του συστήματος ο ApplicationMaster είναι ένα container.
- **YARN container**: Ένα YARN container είναι ένα σύνολο πόρων που έχει δοθεί από τον ResourceManager σε μια επιμέρους διεργασία της εφαρμογής και επιβλέπεται από τον NodeManager του κόμβου στον οποίο τρέχει.

Hadoop MapReduce

Το Hadoop MapReduce βασίζεται στο YARN και αναλαμβάνει την παράλληλη επεξεργασία σε μεγάλα σύνολα δεδομένων. Έχει πάρει την ονομασία του από τους δύο βασικούς αλγορίθμους του, που ονομάζονται “Map” και “Reduce”. Κατά τη διαδικασία του Map ένα σύνολο δεδομένων μετασχηματίζεται σε ένα άλλο, όπου τα στοιχεία σπάνε σε πλειάδες (key/value pairs). Το Reduce λαμβάνει ως είσοδο το αποτέλεσμα ενός Map, το επεξεργάζεται και συνδυάζει της πλειάδες σε ένα μικρότερο σύνολο.



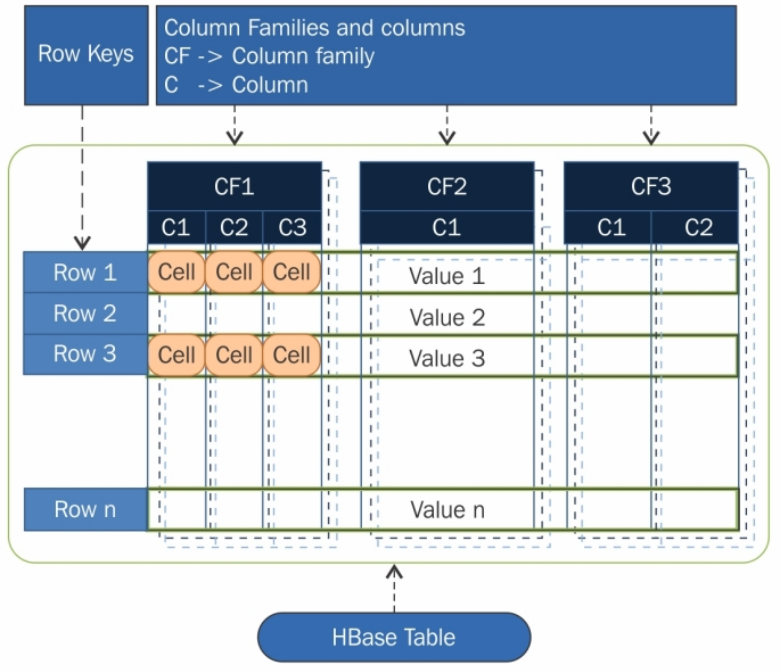
Σχήμα 2.6: Παράδειγμα Map Reduce

2.3.2 Apache HBase

Η Apache HBase [3] είναι μια ανοιχτού κώδικα μη-σχεσιακή βάση δεδομένων σχεδιασμένη σε αντιστοιχία με το Bigtable της Google. Παρέχει τη δυνατότητα αποθήκευσης πολύ μεγάλων πινάκων και την κάνει ιδανική επιλογή για αραιά δεδομένα ή δεδομένα με διαφορετική διάρθρωση, ενώ επιτρέπει την τυχαία και πραγματικού χρόνου πρόσβαση σε αυτά. Η HBase ενσωματώνεται με το Hadoop και δουλεύει απροβλημάτιστα με άλλα εργαλεία πρόσβασης μέσω του YARN. Αποτελεί μέλος της οικογένειας των Apache Projects.

Η HBase διαθέτει τα εξής χαρακτηριστικά:

- Γραμμική και αρθρωτή κλιμάκωση
- Αυστηρή συνέπεια στις αναγνώσεις και εγγραφές
- Αυτόματο και παραμετροποιήσιμο διαχωρισμό των πινάκων καθώς το πλήθος τον δεδομένων αυξάνεται
- Υποστήριξη αυτόματης επαναφοράς λειτουργίας μεταξύ τον RegionServers σε περίπτωση σφάλματος
- Βασικές κλάσεις που διευκολύνουν την εκτέλεση Haddop MapReduce Jobs με πίνακες τις HBase
- Εύκολο στη χρήση Java API για πρόσβαση από το χρήστη
- Διατήρηση των HBase blocks σε λανθάνουσα μνήμη και φίλτρα Bloom για ερωτήματα πραγματικού χρόνου
- Επιτρέπει την εξαγωγή μετρικών μέσω του υποσυστήματος μετρικών του Hadoop (Hadoop metrics subsystem) προς το Ganglia



Σχήμα 2.7: Το μοντέλο δεδομένων στην HBase

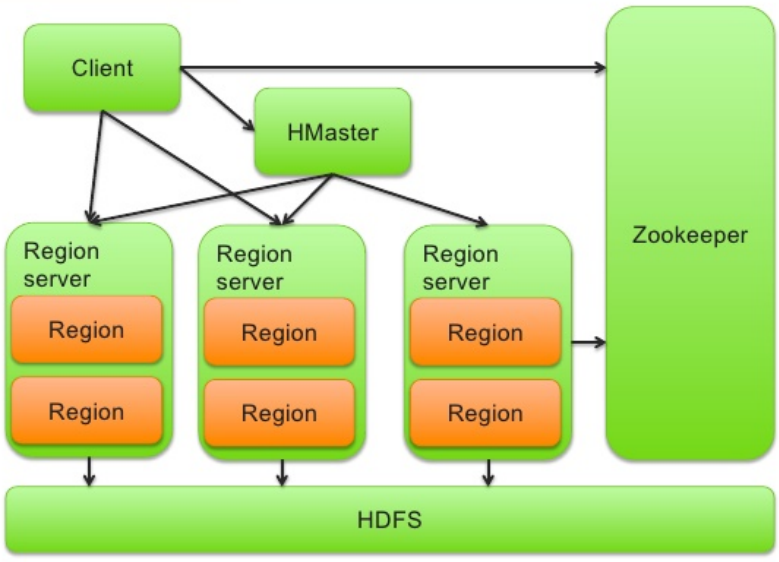
Το μοντέλο δεδομένων της HBase αποτελείται από τα παρακάτω στοιχεία:

- **table**: Αποτελεί το μέσω οργάνωσης των δεδομένων στην HBase
- **row**: Μέσα σε έναν πίνακα (table) τα δεδομένα διατηρούνται σε γραμμές (rows), οι οποίες χαρακτηρίζονται από έναν μοναδικό κλειδί (rowkey) και φιλοξενούν μια η περισσότερες οικογένειες από κολόνες τιμών. Μέσα σε έναν πίνακα οι γραμμές διατηρούνται ταξινομημένες λεξικογραφικά με βάση το κλειδί γραμμής.
- **column**: Μια κολόνα ορίζεται από την οικογένεια κολόνων (column family) που ανήκει και τον ονομαστικό της προσδιορισμό (column qualifier). Αναφορά σε μια κολόνα μπορεί να γίνει με τη μορφή "columnFamily:columnQualifier".
- **column family**: Μια οικογένεια κολόνων ομαδοποιεί τις κολόνες ενός πίνακα και αυτό δε γίνεται μόνο στα μάτια του χρήστη, αλλά και στο σύστημα, καθώς τα μέλη μιας οικογένειας κολόνων αποθηκεύονται μαζί στο σύστημα αρχείων. Οι οικογένειες κολόνων πρέπει να ορίζονται κατά τον ορισμό του σχήματος του πίνακα, και αν και κάθε γραμμή του πίνακα διαθέτει όλες τις οικογένειες κολόνων, δε σημαίνει ότι θα έχουν όλες πάντα δεδομένα.
- **column qualifier**: Αντίθετα με τις οικογένειες κολόνων το ονομαστικό προσδιοριστικό μιας κολόνας δε χρειάζεται να οριστεί κατά τη δημιουργία του σχήματος, με αποτέλεσμα να υπάρχει η δυνατότητα για διαφορετικά column qualifiers από γραμμή σε γραμμή του ίδιου πίνακα.

- **cell**: Ένα κελί είναι ο συνδυασμός γραμμής και κολόνας και σε αυτό αποθηκεύονται τα δεδομένα. Η HBase μας παρέχει τη δυνατότητα να κρατάμε πολλές εκδόσεις (versioning) ενός κελιού.
- **timestamp**: Το timestamp είναι αυτό που χαρακτηρίζει την έκδοση ενός κελιού. Αποθηκεύεται μαζί με το κελί και προσδιορίζεται είτε από το χρήστη είτε αυτόματα από τους RegionServers.

Η HBase υποστηρίζει τις παρακάτω λειτουργίες πάνω στα δεδομένα:

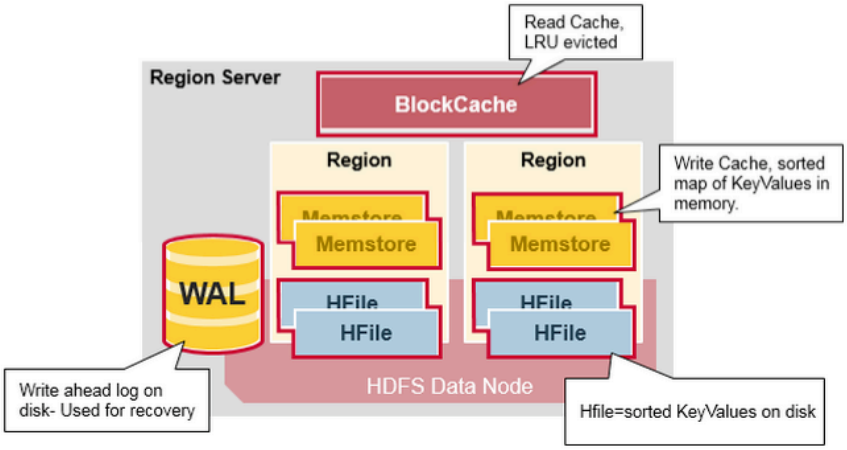
- **Get**: Το Get επιστρέφει χαρακτηριστικά για μια συγκεκριμένη γραμμή.
- **Put**: Το Put είτε προσθέτει νέες γραμμές σε έναν πίνακα (αν δεν υπάρχει το κλειδί) είτε ενημερώνει τις ήδη υπάρχουσες.
- **Scan**: Το Scan επιστρέφει ένα ένα εύρος γραμμών για όλα ή για συγκεκριμένα χαρακτηριστικά. Επίσης μπορούν να οριστούν φίλτρα για τον περιορισμό των αποτελεσμάτων.
- **Delete**: Το Delete διαγράφει μια γραμμή από τον πίνακα. Η HBase, ωστόσο, δεν διαγράφει τα δεδομένα επιτόπου και οι διαγραφές χειρίζονται από τη δημιουργία δεικτών που ονομάζονται "tombstones". Τα tombstones καθαρίζονται μαζί με τις "νεκρές" τιμές κατά τη διάρκεια των μεγάλων συμπιέσεων (HBase Major Compactions).



Σχήμα 2.8: Η αρχιτεκτονική ενός HBase Cluster

Ένα HBase cluster αποτελείται από τρεις τύπους server σε αρχιτεκτονική κύριου/ δευτερεύοντος (master/slave). Τον HMaster, ο οποίος είναι υπεύθυνος για την παρακολούθηση όλων των RegionServers, τις αναθέσεις περιοχών, και για τη δημιουργία/ενημέρωση/διαγραφή

πινάκων. Τους RegionServers, οι οποίοι είναι υπεύθυνοι για την εξυπηρέτηση και τη διαχείριση των περιοχών (regions). Οι RegionServers ζουν μαζί με τα HDFS DataNodes, πράγμα που τους δίνει πιο γρήγορη πρόσβαση στα δεδομένα, αφού βρίσκονται τοπικά (data locality). Τέλος η HBase χρησιμοποιεί το ZooKeeper σαν κατανεμημένη υπηρεσία διοργάνωσης, για να διατηρεί την κατάσταση των server στο cluster. Το ZooKeeper διατηρεί την πληροφορία για το ποιοι servers είναι "ζωντανοί" και διαθέσιμοι, και παρέχει ειδοποιήσεις για αστοχίες.



Σχήμα 2.9: Τα μέρη ενός HBase RegionServer

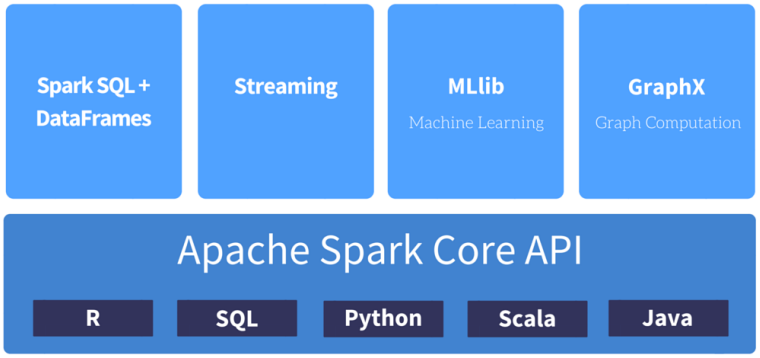
Ένας RegionServer αποτελείται αποτελείται από τα παρακάτω μέρη:

- **WAL**: Το Write Ahead Log είναι ένα αρχείο στο σύστημα αρχείων. Χρησιμοποιείται για την αποθήκευση νέων δεδομένων που δεν έχουν γίνει μόνιμα στο σύστημα, και χρησιμεύει επίσης για την ανάκτηση δεδομένων σε περίπτωση σφάλματος.
- **BlockCache**: Η BlockCache αποτελεί τη λανθάνουσα μνήμη ανάγνωσης. Διατηρεί τακτικά δεδομένα στη μνήμη και χρησιμοποιεί πολιτική τελευταίας χρήσης (Last Recently Used - LRU) στην περίπτωση που γεμίσει.
- **MemStore**: Η MemStore αποτελεί τη λανθάνουσα μνήμα εγγραφής. Διατηρεί νέα δεδομένα που δεν έχουν γραφεί στο δίσκο, τα οποία ταξινομούνται πριν γραφτούν. Αξίζει να σημειωθεί ότι υπάρχει ένα MemStore για κάθε column family για κάθε region.
- **HFiles**: Στα HFiles διατηρούνται οι γραμμές των πινάκων στο δίσκο.

2.3.3 Apache Spark

Το Apache Spark [4] είναι ένα ανοιχτού-κώδικα framework που εκτελεί υπολογισμούς δεδομένων στη μνήμη σε συστάδες υπολογιστών (cluster computing). Η ανάπτυξή του

ξεκίνησε στο AMPLab του University of California, Berkeley και πλέον αποτελεί μέλος της οικογένειας των Apache Projects.
Το Spark έχει γίνει γνωστό για την ταχύτητα των υπολογισμών του όπως και για την κλιμακωσιμότητά του, και λέγεται ότι μπορεί να είναι μέχρι και 100 φορές πιο γρήγορο από το Hadoop MapReduce στη μνήμη και μέχρι και 10 φορές πιο γρήγορο όταν τρέχει στο δίσκο. Έτσι πολλές φορές προσφέρεται ως εναλλακτική στο Hadoop MapReduce. Πάνω σε αυτό έρχεται να δώσει επιπλέον βαρύτητα η δυνατότατη του Spark για ανάγνωση δεδομένων από ένα πλήθος πηγών όπως HDFS, HBase, Cassandra, Hive, Tachyon και οποιαδήποτε άλλη πηγή δεδομένων βασίζεται στο Hadoop.
Το Spark είναι αρκετά ευέλικτο και φιλικό στον προγραμματιστή, καθώς υποστηρίζει τη γρήγορη ανάπτυξη εφαρμογών σε Java, Scala, Python και R. Το framework αυτό διαθέτει τεχνική οκνηρής αποτίμησης (lazy evaluation) πράγμα που του επιτρέπει να είναι αρκετά αποδοτικό στην επεξεργασία μεγάλων συνόλων δεδομένων, καθώς αποθηκεύει μόνο τους μετασχηματισμούς που εκτελεί πάνω σε αυτά και αποτιμά τα δεδομένα μόνο όταν πραγματικά χρειάζεται. Επίσης, το Spark είναι βασισμένο πάνω σε μια δομή δεδομένων που ονομάζεται Resilient Distributed Dataset (RDD), η οποία αποτελεί μια αναπαράσταση ενός συνόλου δεδομένων μόνο για ανάγνωση που είναι διαμοιρασμένη στους κόμβους του Spark Cluster.



Σχήμα 2.10: Τα μέρη του Apache Spark

Το Apache Spark framework αποτελείται από το Spark Core και μια ομάδα βιβλιοθηκών που είναι χτισμένες πάνω σε αυτό.

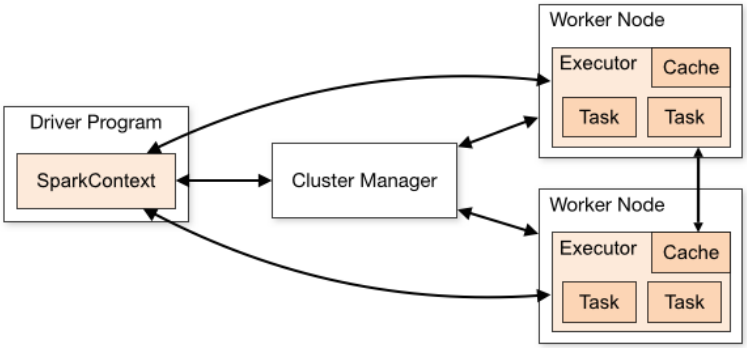
- **Spark Core**: Το Spark Core είναι το θεμέλιο όλου του project. Αποτελεί τη μηχανή κατανεμημένης εκτέλεσης των υπολογισμών και παρέχει τη διανομή των κατανεμημένων εργασιών (distributed task dispatching), χρονοδρομολόγησης και βασικών λειτουργιών εισόδου/εξόδου.

- **Spark SQL**: Η Spark SQL είναι μια βιβλιοθήκη που παρέχει υποστήριξη στο Spark για δομημένα και ημι-δομημένα δεδομένα. Επίσης δίνει του δίνει τη δυνατότητα να εκτελέσει ερωτήματα τύπου SQL και εισάγει δύο αφαιρετικές δομές, το Dataset και το Dataframe.

- **Spark Streaming**: Η Spark Streaming είναι μια βιβλιοθήκη που δίνει στο Spark

τη δυνατότητα επεξεργασίας δεδομένων σε πραγματικό χρόνο. Σε αντίθεση με άλλα frameworks (Apache Storm, Apache Flink) το spark δε διατηρεί μια συνεχή ροή δεδομένων, αλλά βασίζεται σε μια mini-batch μορφή επεξεργασίας και υπολογισμού.

- **Spark MLlib**: Η Spark MLlib αποτελεί μια κλιμακώσιμη βιβλιοθήκη για μηχανική μάθηση που περιέχει αλγόριθμους και εργαλεία για classification, regression, clustering, dimensionality reduction κα.
- **Spark GraphX**: Η Spark GraphX είναι μια βιβλιοθήκη που επεκτείνει το Spark και παρέχει τη δυνατότητα υπολογισμών σε γράφους.



Σχήμα 2.11: Η αρχιτεκτονική του Spark

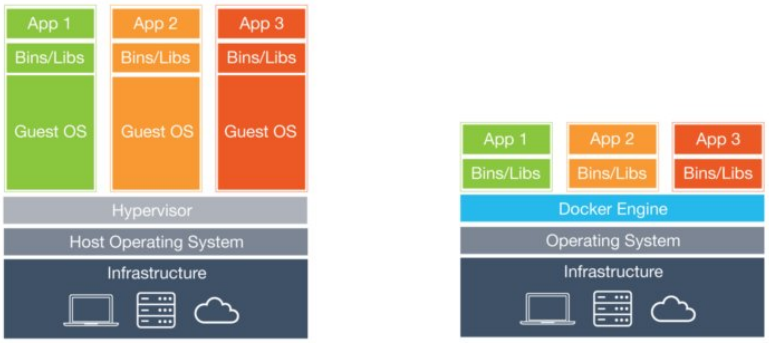
Η αρχιτεκτονική μιας εφαρμογής Spark αποτελείται από της εξής οντότητες:

- **Application Jar**: Είναι ένα Jar που περιέχει το πρόγραμμα του χρήστη για το Spark. Πολλές φορές το Jar αυτό περιλαμβάνει επίσης βιβλιοθήκες στις οποίες βασίζεται το πρόγραμμα του χρήστη.
- **Driver Program**: Το Driver program είναι η διεργασία που τρέχει στον Master κόμβο και έχει τη main μέθοδο της εφαρμογής. Εκεί δημιουργείται επίσης και το SparkContext.
- **Cluster Manager**: Ο Cluster Manager είναι αυτός που διαχειρίζεται τους πόρους του cluster. Το Spark παρέχει τον δικό του cluster manager για την περίπτωση που θέλουμε να τρέξει σε Standalone Mode, όμως μπορεί να δουλέψει επίσης με το Apache Mesos, Hadoop YARN και Kubernetes(πειραματικά).
- **Worker Node**: Worker Node μπορεί να χαρακτηριστεί οποιοσδήποτε κόμβος στο cluster που μπορεί να εκτελέσει κώδικα της εφαρμογής.
- **Executor**: Executor ονομάζεται μια διεργασία που εκκινείται για μια εφαρμογή σε κάποιο Worker Node, εκτελεί διάφορα κομμάτια εργασίας και κρατάει τα δεδομένα στη μνήμη ή στο δίσκο.

- **Task**: Task ονομάζεται ένα κομμάτι εργασίας που αποστέλλεται σε έναν executor για εκτέλεση.
- **Job**: Job ονομάζεται ένας παράλληλος υπολογισμός που αποτελείται από πολλά tasks.
- **Stage**: Κάθε Job διαιρείται σε μικρότερα σύνολα από tasks, αυτά ονομάζονται stages που εξαρτώνται το ένα από το άλλο.

2.3.4 Docker

Το Docker [11] είναι μια πλατφόρμα για τη δημιουργία, διαχείριση και ανάπτυξη containers λογισμικού. Υποστηρίζει τόσο Windows όσο και Linux συστήματα.
Το Docker είναι ένα εργαλείο που ωφελεί τόσο τους προγραμματιστές όσο και τους διαχειριστές συστημάτων. Για τους προγραμματιστές, αφαιρεί το βάρος της παραμετροποίησης κάθε συστήματος, αφού τους βοηθάει να αναπτύσσουν εφαρμογές ανεξάρτητες από το περιβάλλον στο οποίο θα τρέξουν. Έτσι, τους επιτρέπει να επικεντρώνονται στην εφαρμογή και επιταχύνει την διαδικασία παράδοσης και ενεργοποίησης νέων χαρακτηριστικών για εφαρμογές. Για τους διαχειριστές συστημάτων (system administrators) το Docker, δίνει ευελιξία και πιθανότητα μειώνει τα συστήματα που χρειάζονται.



Σχήμα 2.12: Virtual Machines vs Containers

Τα containers είναι αυτόνομα περιβάλλοντα εκτέλεσης και διαθέτουν μεμονωμένους πόρους επεξεργαστή, μνήμης, εισόδου/εξόδου στο δίσκο και δικτύου.
Σε αντίθεση με τα Virtual Machines, τα containers είναι πιο ελαφριά και επιβαρύνουν πολύ λιγότερο το μηχάνημα στο οποίο τρέχουν. Αυτό συμβαίνει γιατί μοιράζονται τον kernel του μηχανήματος που τα φιλοξενεί και δεν δημιουργούν ένα ολόκληρο λειτουργικό σύστημα.

Τέλος το Docker μέσω του Docker Hub προσφέρει μια πολύ μεγάλη βιβλιοθήκη από έτοιμα containers, υλοποιημένα από την κοινότητα του Docker, που μπορεί να πάρει κάποιος και να χτίσει τα δικά του πάνω σε αυτά. Έτσι δε απαιτείται η δημιουργία ενός νέου container από το μηδέν. Στο Docker Hub μπορεί να βρει κανείς containers για Wordpress, Mongo DB, MySQL, ElasticSearch, Hadoop και άλλα πολλά δημοφιλή frameworks.

2.3.5 HiBench

Το HiBench [10] είναι μια σουίτα μετροπρογραμμάτων για μεγάλα δεδομένα (Big Data) [28]. Έχει σχεδιαστεί για να βοηθάει στην αποτίμηση και την παραμετροποίηση διαφόρων big data frameworks, σε σχέση με τη ταχύτητα (speed), απόδοση (throughput) και χρησιμοποίησης πόρων συστήματος (system resource utilization). Το HiBench περιλαμβάνει κυρίως φόρτους εργασίας (Workloads) για Hadoop και Spark, ενώ τελευταία έχουν αρχίσει να προστίθενται και φόρτοι εργασίας ροών δεδομένων (streaming workloads) που στοχεύουν Spark Streaming, Flink, Storm και Gearpump.

Συνολικά υπάρχουν 19 φόρτοι εργασίας στο HiBench που μπορούν να χωριστούν σε 6 κατηγορίες.

Micro Benchmarks

Sort: Αυτό το workload λαμβάνει ως είσοδο κείμενο, που παράγεται από τον RandomTextWriter, και το ταξινομεί.

WordCount: Αυτό το workload λαμβάνει ως είσοδο κείμενο, που παράγεται από τον RandomTextWriter, και μετράει τις εμφανίσεις κάθε λέξης μέσα σε αυτό.

TeraSort: Το Terasort είναι ένα πρότυπο μετροπρόγραμμα που έχει δημιουργηθεί από τον Jim Gray, το οποίο μετράει το χρόνο που χρειάζεται ένα σύστημα για να ταξινομήσει ένα Terabyte κατανεμημένων δεδομένων.

Sleep: Αυτό το workload δοκιμάζει την απόδοση του χρονοδρομολογητή του κάθε framework, "κοιμίζοντας" κάθε διεργασία για ένα συγκεκριμένο αριθμό δευτερολέπτων.

Enhanced DFSIO: Αυτό το workload δοκιμάζει την απόδοση του HDFS στο Hadoop cluster, δημιουργώντας μεγάλο αριθμό διεργασιών που εκτελούν εγγραφές και αναγνώσεις ταυτόχρονα. Αξιολογεί το μέσο ρυθμό εισόδου/εξόδου για κάθε Map, τη μέση απόδοση για κάθε Map και τη συνολική απόδοση του HDFS cluster. Σημειώνεται ότι το συγκεκριμένο workload δεν έχει υλοποίηση σε Spark.

Machine Learning

Bayesian Classification: Αυτό το wokrload αξιολογεί την επίδοση του NaiveBayesian Clissification βασισμένο στη Spark-MLlib και το Mahout. Για την εκτέλεσή του, το workload αυτό δημιουργεί αυτόματα αρχεία των οποίων οι λέξης ακολουθούν zipfian κατανομή. Το λεξικό που χρησιμοποιείται είναι το προκαθορισμένο αρχείο στα Linux που βρίσκονται στην θέση /usr/share/dict/linux.words.

K-means clustering: Αυτό το workload αξιολογεί την επίδοση του K-means clustering, βασισμένο στη Spark-MLlib και το Mahout. Το σύνολο δεδομένων στο οποίο εκτελείται, είναι βασισμένο σε ομοιόμορφη κατανομή (Uniform Distribution) και γκαουσιανή κατανομή (Guassian Distribution).

Logistic Regression: Αυτό το workload δοκιμάζει την απόδοση του αλγορίθμου Logistic Regression βασισμένου στη Spark-MLlib και υλοποιείται με την αριθμητική τεχνική βελτιστοποίησης LBFGS. Το σύνολο δεδομένων, πάνω στο οποίο εκτελείται ο αλγόριθμος, περιέχει τρία διαφορετικά είδη τύπων δεδομένων. Συμπεριλαμβάνει κατηγορικά δεδομένα, συνεχή δεδομένα και δυαδικών δεδομένων.

Alternating Least Squares: Αυτό το workload δοκιμάζει την απόδοση του αλγορίθμου Alternating Least Squares (Εναλλασσόμενα Ελάχιστα Τετράγωνα) βασισμένου στη Spark-MLlib. Τα δεδομένα εισόδου προέρχονται από ένα σύστημα προτάσεων προϊόντων.

SQL

Scan, **Join**, **Aggregate**: Αυτά τα workloads είναι βασισμένα στη δημοσίευση "A Comparison of Approaches to Large-scale Data Analysis" [21] και στο νήμα ανάπτυξης του HIVE, *Hive-396: Hive performance benchmarks* [9].

Websearch Benchmarks

PageRank: Αυτό το workload αξιολογεί την απόδοση του αλγορίθμου PageRank βασισμένου στην Spark-MLlib και στο Hadoop. Τα δεδομένα εισόδου προέρχονται από δεδομένα ιστού (web data), των οποίων οι υπερ-σύνδεσμοι ακολουθούν zipfian κατανομή.

Nutch Indexing: Αυτό το workload αξιολογεί την απόδοση του υποσυστήματος ευρετηρίου του Nutch, μιας ανοιχτού κώδικα μηχανής αναζήτησης που ανήκει στην οικογένεια των Apache Projects. Τα δεδομένα εισόδου προέρχονται από δεδομένα ιστού (web data), των οποίων οι υπερ-σύνδεσμοι και οι λέξεις ακολουθούν zipfian κατανομή.

Graph Benchmarks

NWeight: Το workload αυτό υπολογίζει συσχετισμούς μεταξύ δύο κορυφών που απέχουν απόσταση N. Είναι ένας επαναληπτικός παράλληλος αλγόριθμος βασισμένος στη Spark GraphX και το pregel.

Streaming Benchmarks

Identity: Το workload αυτό διαβάζει δεδομένα από το Kafka και γράφει αμέσως τα αποτελέσματα στο Kafka.

Repartition: Το workload αυτό διαβάζει δεδομένα από το Kafka και αλλάζει το βαθμό παραλληλισμού δημιουργώντας λιγότερες ή περισσότερες δοκιμές διαχωρισμού. Δοκιμάζει την αποδοτικότητα των streaming frameworks να ανακατανέμουν τα δεδομένα.

Stateful Wordcount: Το workload αυτό μετράει λέξης που λήφθηκαν συνολικά από το Kafka κάθε μερικά δευτερόλεπτα. Δοκιμάζει την απόδοση των streaming frameworks διατηρώντας μια κατάσταση και το κόστος που προσθέτουν τα σημεία ελέγχου (Checkpoints).

Fixwindow: Αυτά τα workloads δοκιμάζουν την απόδοση των streaming frameworks σε πράξεις παραθύρων.

Κεφάλαιο 3

Εύρεση Ανωμαλιών

Για τον εντοπισμό ανωμαλιών σε χρονοσειρές με περιοδικότητα, έχουν προταθεί αρκετές τεχνικές, όπως Seasonal Trend Decomposition [8], Classification και Regression [16].
Το Principal Component Pursuit στον Robust PCA έχει ως συνέπεια της αποδόμησης πινάκων, τον εντοπισμό σημείων που διαφέρουν πολύ απο τα υπόλοιπα (outliers) και για αυτή του την ιδιότητα έχει αρχίσει να εφαρμόζεται σαν μέθοδος εντοπισμού ανωμαλιών [1, 12].
Η εύρεση ανωμαλιών μέσω του PCP έχει χρησιμοποιηθεί επίσης από το Netflix, το οποίο έχει δημιουργήσει και μια βιβλιοθήκη ανοιχτού κώδικα στην πλατφρόμα GitHub, που παρέχει μια υλοποίηση σε Java [23, 27].

Στη συνέχεια παρουσιάζουμε πως χρησιμοποιήσαμε την τεχνική του RobustPCA για εντοπισμό ανωμαλιών στις χρονοσειρές από τους υπολογιστικούς κόμβους.

3.1 Αλγόριθμος

Στον αλγόριθμο 1 περιγράφουμε τον τρόπο με τον οποίο εντοπίζουμε τις ανωμαλίες. Ο αλγόριθμος δέχεται ως είσοδο τρεις παραμέτρους:

- F: Η εποχικότητα που θα εφαρμοστεί στη μελέτη
- H: Σύνολο ιστορικών σημείων, με πλήθος που διαιρείται με το F
- C: Σύνολο νέων σημείων, με πλήθος που διαιρείται με το F

Για να γίνει η ανάλυση είναι σημαντικό στα δύο σύνολα δεδομένων H και C, όταν χωρίζονται με την εποχικότητα F, τα σημεία αρχής και τέλους να βρίσκονται αρκετά κοντά μεταξύ τους. Παραδείγματος χάρη, αν η εποχικότητα μας είναι ημερήσια και ο κύκλος ξεκινάει στις 00:00 και τελειώνει στις 23:59, τότε τα επιμέρους τμήματα των συνόλων H και C πρέπει να αρχίζουν και να τελειώνουν πολύ κοντά σε αυτές τις χρονικές στιγμές.
Επιπλεόν, για την ανακάλυψη της εποχικότητας μπορούν να χρησιμοποιηθούν στατιστικές

μέθοδοι όπως ο συντελεστής αυτοσυσχέτισης (Autocorellation Factor - ACF) και ο μερικός συντελεστής αυτοσυσχέτισης (Partial Autocorellation Factor - PACF) [5].

Ο αλγόριθμος 1 μπορεί να περιγραφεί ως εξής:
Κατά την ανάλυση των δεδομένων, ενοποιούμε τα δύο σύνολα και κανονικοποιούμε τα δεδομένα. Έπειτα αναδιατάσσουμε τη χρονοσειρά με τα σημεία σε δισδιάστατη μορφή βάσει της εποχικότητας που έχουμε θέσει. Εκτελούμε τον αλγόριθμο Robust PCA και λαμβάνουμε δύο δισδιάστατους πίνακες, τον L που αποτελεί τη συνιστώσα χαμηλής τάξης των δεδομένων και τον S που αποτελεί την αραιά συνιστώσα με τις ανωμαλίες. Ασχέτως με το αν ο αρχικός πίνακας είχε αρνητικές τιμές ή όχι, οι πίνακες L και S μετά την ανάλυση είναι δυνατόν να περιέχουν αρνητικές τιμές. Έτσι παίρνουμε το απόλυτο των τιμών του πίνακα S, ώστε να μπορέσουμε να δημιουργήσουμε μια κατανομή και να επιλέξουμε τις πιο σημαντικές ανωμαλίες. Στη συνέχεια φιλτράρουμε τις ανωμαλίες και διατηρούμε αυτές που βρίσκονται στο παράθυρο που μας ενδιαφέρει.

Algorithm 1: Εντοπισμός Ανωμαλιών Σε Υπολογιστικό Κόμβο

Input: H, C, F
1 U = $H \cup C$;
2 Κανονικοποίηση των δεδομένων;
3 Μετατροπή σε δισδιάστατο πίνακα βάσει του F;
4 L, S = RobustPCA();
5 Εφαρμογή απόλυτης τιμής στις τιμές του S;
6 TopX = Εύρεση του Top X% των τιμών του S;
7 Επιστροφή των σημείων που βρίσκονται στην τομή $TopX \cap C$;

Η χρήση του RobustPCA και της τεχνικής του για την αποδόμηση του πίνακα των μετρήσεων [7], χρησιμοποιείται ως μαύρο κουτί από τον αλγόριθμο 1. Στις δοκιμές που διεξάγαμε ωστόσο η μέθοδος για την αποδόμηση υλοποιήθηκε εξ' αρχής τόσο σε Python όσο και σε Scala A′.1, ως μεταφορά μιας πρότυπης υλοποίησης σε Matlab.

Όπως αναφέρθηκε στο κεφάλαιο 2 υπάρχουν αρκετοί τρόποι που μπορεί να υλοποιηθεί ο αλγόριθμος αναζήτησης των κύριων συνιστωσών (Principal Component Pursuit) [17, 13, 30, 6]. Επιλέχθηκε να γίνει η υλοποίηση με τη μέθοδο των Πολλαπλασιαστών Εναλλασσόμενων Κατευθύνσεων (Alternating Direction Method of Multiplier - ADMM). Αυτή η επιλογή έγινε γιατί η μέθοδος ADMM παρουσιάζει πιο γρήγορη σύγκλιση θυσιάζοντας λίγο περισσότερο το σφάλμα απο τις άλλες μεθόδους, αλλά και γιατί φαινόταν πιο υποσχόμενη τακτική για μια παράλληλη υλοποίηση [19, 20].

Η πρότυπη υλοποίηση που χρησιμοποιήθηκε για την αποδόμηση του πίνακα των μετρήσεων βρίσκεται στα παραδείγματα για προσεγγιστικούς αλγόριθμους των N. Parikh και S. Boyd [18].

Κεφάλαιο 4

Σύνολα Δεδομένων (Data sets)

4.1 Δημιουργία Custom Dataset

Για να μπορέσουμε να αποτιμήσουμε την ακρίβεια της τεχνικής που εφαρμόζουμε, χρειαζόμαστε ένα σύνολο δεδομένων στο οποίο να γνωρίζουμε που έχουν συμβεί ανωμαλίες. Ωστόσο, αν και υπήρχαν αρκετά σύνολα δεδομένων παρακολούθησης πόρων συστάδων υπολογιστών, κανένα από τα δημόσια διαθέσιμα σύνολα δεδομένων δεν περιείχε την πληροφορία που ψάχναμε. Έτσι οδηγηθήκαμε να δημιουργήσουμε ένα δικό μας σύνολο δεδομένων, παρακολουθώντας μια συστάδα υπολογιστών, στην οποία εκτελούσαμε ανά συγκεκριμένα χρονικά διαστήματα (4 ώρες) ένα σύνολο από workloads της σουίτας HiBench για Hadoop και Spark.

4.1.1 Cluster Setup

Το cluster μας αποτελείται από 5 μηχανήματα τα οποία δημιουργήθηκαν στο ιδιωτικό υπολογιστικό νέφος του εργαστηρίου, βασισμένο στο OpenStack, και διαθέτουν τα εξής χαρακτηριστικά:

- OS: Ubuntu 16.04
- CPU: 4 virtual cores
- RAM: 4Gb
- HDD: 60Gb

Επιπλέον δημιουργήσαμε ένα ακόμη Volume των 20Gb στο οποίο γράφονται οι μετρήσεις από το Ganglia, και όπως θα δούμε παρακάτω για τη δημιουργία του cluster χρησιμοποιήσαμε Docker Containers.
Για να μπορούμε να αναφερθούμε εύκολα στη συνέχεια στους κόμβους του cluster, τους ονομάζουμε Master, Slave 1, Slave 2, Slave 3, Slave 4.

Παραμετροποίηση Ganglia

Παραχωρήσαμε το Volume των 20Gb στον Master και οι δαίμονες του συστήματος παρακολούθησης Ganglia εγκαταστάθηκαν στο Ubuntu 16.04 που φιλοξενούσε τα containers. Η παραμετροποίηση του Ganglia έγινε ως εξής:

- Ο Master ανέλαβε το καθήκον εκτέλεσης του Ganglia Meta Daemon (gmetad). Αλλάξαμε τον προκαθορισμένο φάκελο αποθήκευσης της βάσης δεδομένων του Ganglia και ρυθμίσαμε τη βάση να κρατάει δύο Round Robin Archives (RRA) για κάθε τύπο μέτρησης και για κάθε κόμβο του cluster. Το πρώτο RRA διατηρούσε μετρήσεις 10 δευτερολέπτων για διάστημα 2 εβδομάδων ενώ το δεύτερο διατηρούσε μετρήσεις 5 λεπτών για διάστημα ενός μήνα. Το gmetad ρυθμίστηκε να κάνει poll για δεδομένα κάθε 2 δευτερόλεπτα και τα RRAs αποθήκευαν το μέσο όρο για κάθε 10 δευτερόλεπτα ή 5 λεπτά αντίστοιχα.

- Σε κάθε κόμβο εκτελούταν το Ganglia Monitor Daemon (gmond), το οποίο είχε παραμετροποιηθεί έτσι ώστε έχει νόημα το γρήγορο polling του gmetad από τον Master. Δηλαδή ο χρόνος που μεσολαβούσε για κάθε μέτρηση της χρησιμοποίησης των πόρων είχε μειωθεί αρκετά και για πόρους που μπορούσαμε να έχουμε ραγδαία μεταβολή (πχ. cpu, ram) είχε τεθεί στα 2 δευτερόλεπτα. Επιπλέον ενεργοποιήσαμε το diskstat module, γραμμένο σε python, που επιτρέπει στο gmond να λαμβάνει πιο λεπτομερείς μετρήσεις για το δίσκο.

Docker Images

Για τη δημιουργία του cluster χρησιμοποιήσαμε Docker containers με εικόνες (images) που χτίσαμε για τις ανάγκες μας. Επειδή όλοι οι κόμβοι απαιτούν την ύπαρξη ενός προκαθορισμένου περιβάλλοντος παραμετροποιημένο για Hadoop και Spark, δημιουργήσαμε ένα βασικό (base) Docker image που περιλάμβανε τα εξής βασικά πακέτα και frameworks:

Πίνακας 4.1: Docker Base Image Packages

Όνομα Πακέτου	Έκδοση
Python	2.7.3
JDK	8
Maven	3.5.0
Scala	2.11
Hadoop	2.8.0
Spark	2.1.1

Στη συνέχεια από αυτό το base Docker image δημιουργήσαμε τις εικόνες για τον Master και τους Slaves. Η εικόνα των Slave κόμβων δεν περιλαμβάνει πολλές αλλαγές, καθώς φορτώνουμε απλά τα αρχεία παραμετροποίησης των frameworks και εκθέτουμε (expose) τις πόρτες που θα χρησιμοποιηθούν από τα frameworks.
Αντίθετα στην εικόνα του container για τον Master, πέρα από τη φόρτωση των αρχείων

παραμετροποίησης των frameworks, κατεβάζουμε και χτίζουμε την τελευταία έκδοση του HiBench (6.0) για το περιβάλλον εκτέλεσής μας, εισάγουμε κατάλληλη εγγραφή στο εργαλείο cron για την αυτόματη εκτέλεση των workloads και δημιουργούμε entrypoint script (σενάριο εισόδου) που δημιουργεί το cluster κάθε φορά που εκκινεί το container.

Το entrypoint script είναι το ακόλουθο:

Script 4.1: HiBench Master Entrypoint Script

```
#!/bin/bash

/usr/sbin/sshd
/root/start_cluster.sh
/usr/sbin/cron

kill -9 $(cat /var/run/sshd.pid)
exec /usr/sbin/sshd -D
```

Script 4.2: HiBench Cluster Start Script

```
#!/bin/bash
ssh hibench-slave-1 << EOF
    $HADOOP_PREFIX/bin/hdfs namenode -format hibench
    $HADOOP_PREFIX/sbin/start-dfs.sh
EOF

ssh hibench-slave-2 << EOF
    $HADOOP_PREFIX/sbin/start-yarn.sh
EOF

ssh hibench-slave-3 << EOF
    $HADOOP_PREFIX/sbin/mr-jobhistory-daemon.sh \
        --config $HADOOP_CONF_DIR start historyserver
EOF
```

Όπως φαίνεται από το script ο Master συνδέεται με ssh στους κόμβους και ενεργοποιεί τον Namenode του HDFS cluster στον Slave 1, το Hadoop YARN στον Slave 2 και το Hadoop JobHistory Server στον Slave 3. Αυτό έγινε για να κατανείμουμε το φόρτο του cluster καλύτερα, καθώς χρησιμοποιούσαμε όλους τους κόμβους του για υπολογισμούς.

Παραμετροποίηση YARN, MapReduce, Spark

Στο YARN ορίσαμε ότι ο κάθε κόμβος έχει διαθέσιμα 3 cpu-vcores και 3Gb RAM και αφήσαμε διαθέσιμα στο λειτουργικό των κόμβων 1 cpu-vcore και 1Gb RAM.
Ορίσαμε το Spark σε yarn-client mode και θέσαμε το πλήθος των executors σε 4 και το πλήθος των πυρήνων τον executors σε 3. Επίσης θέσαμε τη μνήμη των executos και του driver σε 1.5Gb, και το μέγεθος του παραλληλισμού στο Spark σε 48.
Σημείωση: Ο αριθμός 48 δεν είναι τυχαίος. Σύμφωνα με τους διατηρητές του HiBench ο παραλληλισμός στο Spark πρέπει να είναι 4-5 φορές μεγαλύτερος από το πλήθος των executors επί τους πυρήνες τους, στην περίπτωσή μας 12.
Τέλος ορίσαμε τη μνήμη που απαιτεί ένα Map Task σε 1Gb και τη μνήμη που απαιτεί ένα Reduce Task σε 2Gb.

HiBench Runs

Από τη σουίτα του HiBench επιλέξαμε να εκτελούμε τα παρακάτω workloads κάθε φορά με την ίδια σειρά.

1. Sleep (Hadoop, Spark)
2. Sort (Hadoop, Spark)
3. Terasort (Hadoop, Spark)
4. WordCount (Hadoop, Spark)
5. DFSIOE (Hadoop)
6. Aggregation (Hadoop, Spark)
7. Join (Hadoop, Spark)
8. Scan (Hadoop, Spark)
9. Pagerank (Hadoop, Spark)
10. Bayesian Classification (Hadoop, Spark)
11. K-means Clustering (Hadoop, Spark)
12. Alternating Least Squares (Spark)

Κατά τη δημιουργία του Docker image για τον Master αναφέραμε ότι εισάγουμε κατάλληλη εγγραφή στο εργαλείο cron για την αυτόματη εκτέλεση των workloads του HiBench. Το cron είναι ένα εργαλείο που επιτρέπει την αυτόματη εκτέλεση διεργασιών στο παρασκήνιο, ανά συγκεκριμένα παραμετροποιήσιμα χρονικά διαστήματα. Το cron διαβάζει το πρόγραμμα εκτέλεσης που πρέπει να ακολουθήσει, από ένα αρχείο που ονομάζεται crontab. Η εγγραφή που εισάγαμε στο crontab εμείς μοιάζει με την παρακάτω

```
0 */4 * * * run_hibench_loop.sh
```

και ορίζει την εκτέλεση των HiBench workloads ανά 4 ώρες καθημερινά.

4.1.2 Εισαγωγή Ανωμαλιών

Κατά τη διάρκεια εκτέλεσης των workloads επιλέξαμε να εισάγουμε στοχευμένα ανωμαλίες στον Slave-3 σε συγκεκριμένες στιγμές. Οι ανωμαλίες που εισάγαμε ήταν τύπου επεξεργαστή, μνήμης, δίσκου, δικτύου και πλήθους ενεργών διεργασιών.

Για την εισαγωγή των ανωμαλιών χρησιμοποιήσαμε το εργαλείο stress, το εργαλείο wget, κάποια scripts που φτιάξαμε για τον περιορισμό της ταχύτητας του δικτύου και ένα πρόγραμμα σε C για τη δημιουργία διεργασιών.

Το stress είναι μια απλή γεννήτρια φόρτου για POSIX συστήματα, η οποία μπορεί να δημιουργήσει παραμετροποιήσιμο τεχνητό φόρτο στον επεξεργαστή, στη μνήμη, στο I/O και στο δίσκο του συστήματος. Είναι υλοποιημένο σε C και αποτελεί δωρεάν λογισμικό με άδεια GPLv2.
Έτσι για να εισάγουμε ανωμαλίες στη χρήση του επεξεργαστή, της μνήμης, I/O και του δίσκου στον Slave-3 χρησιμοποιήσαμε τις παρακάτω εντολές.

```
# Χρήση 3 πυρήνων για 180 δευτερόλεπτα
stress -c 3 -t 180
# Δημιουργία 3 διεργασιών που καταναλώνουν 512Mbyte μνήμης για 120 δευτερόλεπτα
stress -m 3 –vm-bytes 512M –vm-keep -t 120
# Δημιουργία 3 διεργασιών που γράφουν 1Gb στο δίσκο για 300 δευτερόλεπτα
stress -d 3 –hdd-bytes 1G -t 300
```

Το wget είναι ένα δωρεάν εργαλείο που χρησιμοποιείται για την ανάκτηση αρχείων χρησιμοποιώντας πρωτόκολλα HTTP, HTTPS και FTP. Είναι ένα μη διαδραστικό εργαλείο γραμμής εντολών που μπορεί να χρησιμοποιηθεί πολύ εύκολα μέσα από scripts, εργασίες cron και τερματικά.
Το wget το χρησιμοποιήσαμε για να αυξήσουμε τον όγκο το δεδομένων που λάμβανε το σύστημα σε ορισμένες χρονικές στιγμές, κατεβάζοντας μέσω FTP την τελευταία έκδοση του λειτουργικού συστήματος Debian. Ωστόσο για να αποφύγουμε τις εγγραφή του αρχείου στο δίσκο, κάναμε ανακατεύθυνση της εξόδου στην ειδική συσκευή "Null" του συστήματος. Η εντολή που χρησιμοποιήσαμε είναι η παρακάτω:

```
wget -O /dev/null ftp://ftp.ntua.gr/pub/linux/debian-cd/8.8.0/amd64/iso-dvd/debian-8.8.0-amd64-DVD-1.iso
```

Επιπλέον πέρα από την αύξηση των πακέτων στο δίκτυο μέσω του wget υλοποιήσαμε κάποια scripts για να περιορίσουμε την ταχύτητα δικτύου του Slave-3. Αυτό επιτεύχθηκε με τη χρήση του εργαλείου tc και την υλοποίηση ουρών με συγκεκριμένη συμπεριφορά.

Το tc [14] χρησιμοποιείται για την ρύθμιση του ελέγχου κίνησης (Traffic Control) πακέτων δικτύου στον Linux Kernel. Ο έλεγχος περιλαμβάνει τα εξής:

- **Shaping** (Διαμόρφωση): Όταν λέμε ότι γίνεται διαμόρφωση στην κίνηση δικτύου, τότε εννοούμε ότι ο ρυθμός μετάδοσης ελέγχεται και δεν είναι όσο ορίζει το υλικό του συστήματος. Διαμόρφωση μπορεί να έχουμε είτε γιατί θέλουμε να μειώσουμε το εύρος ζώνης είτε γιατί θέλουμε να ομαλοποιήσουμε τις απότομες αυξήσεις στη ροή δεδομένων (bursts in traffic), για καλύτερη συμπεριφορά του δικτύου.

- **Scheduling** (Χρονοδρομολόγηση): Μέσω τις χρονοδρομολόγησης η αποστολή πακέτων μπορεί να βοηθήσει στην καλύτερη διαδραστικότητα για δικτυακή κίνηση που το χρειάζεται, και παράλληλα να εγγυάται εύρος ζώνης για ογκώδη μεταφορές.

- **Policing** (Επιτήρηση): Αντίθετα με τη διαμόρφωση η επιτήρηση γίνεται στα πακέτα τα οποία φτάνουν από το δίκτυο.

- **Dropping** (Απόρριψη): Η κίνηση δικτύου που ξεπερνάει ένα ορισμένο εύρος ζώνης μπορεί να οδηγήσει στην απόρριψη πακέτων, τόσο εξερχόμενων όσο και εισερχόμενων.

Για τον έλεγχο της κίνησης δικτύου χρησιμοποιούνται τρία αντικείμενα:

- **Qdiscs**: Το qdisc αποτελεί συντομογραφία του "Queueing Discipline" (Συμπεριφορά Ουράς), και αποτελεί βασικό στοιχείο για την κατανόηση του ελέγχου κίνησης δικτύου. Κάθε φορά που ο kernel θέλει να στείλει ένα πακέτο σε κάποια διεπαφή (interface) δικτύου, αυτό μπαίνει σε συγκεκριμένη ουρά για τη διεπαφή, ελεγχόμενη από qdisc. Στη συνέχεια ο kernel προσπαθεί να στείλει όσο περισσότερα πακέτα μπορεί από το qdisc στον οδηγό δικτύου.
  Ένα απλό Qdisk είναι μια ουρά pfifo (pure First In, First Out).

- **Classes** (Κλάσεις): Μερικά qdisks μπορούν να αποκτήσουν κλάσεις μέσα στις οποία υπάρχουν άλλα qdisks και η κίνηση δικτύου μπορεί να μπει στην ουρά οπουδήποτε εσωτερικού qdisk. Μέσω τον κλάσεων μπορεί να οριστεί και προτεραιότητα συγκεκριμένων πακέτων.

- **Filters** (Φίλτρα): Τα φίλτρα χρησιμοποιούνται από qdisks που περιέχουν κλάσεις, και ορίζει σε ποια κλάση θα αντιστοιχιστεί κάθε πακέτο.

Για τον περιορισμό της εξερχόμενης κίνησης στον Slave-3 χρησιμοποιήσαμε το παρακάτω script.

Script 4.3: Traffic Shaping Slave-3 OUT

```
#!/bin/bash

interface="enp0s3"
bandwidth="10"
sleep_time="180"

tc qdisc add dev ${interface} root handle 1: htb default 11
tc class add dev ${interface} parent 1: classid 1:1 htb \
    rate ${bandwidth}mbit
tc class add dev ${interface} parent 1:1 classid 1:11 htb \
    rate ${bandwidth}mbit
tc qdisc add dev ${interface} parent 1:11 handle 20: sfq perturb 10

sleep ${sleep_time}
tc qdisc del dev ${interface} root

exit 0
```

Για τον περιορισμό της εισερχόμενης κίνησης προς τον Slave-3 εγκαταστήσαμε και τρέξαμε ένα παρόμοιο script σε όλους τους άλλους κόμβους, που περιόριζε την εξερχόμενη κίνηση από αυτούς προς την διεύθυνση του Slave-3.

Script 4.4: Traffic Shaping Slave-3 IN

```
#!/bin/bash

interface="enp0s3"
bandwidth="10"
max_bandwidth="1000"
destination="192.168.5.64"
sleep_time="180"

tc qdisc add dev ${interface} root handle 1: htb default 11
tc class add dev ${interface} parent 1: classid 1:1 htb \
    rate ${max_bandwidth}mbit
tc class add dev ${interface} parent 1:1 classid 1:10 htb \
    rate ${bandwidth}mbit
tc class add dev ${interface} parent 1:1 classid 1:11 htb \
    rate ${max_bandwidth}mbit

tc filter add dev ${interface} protocol ip parent 1:0 prio 1 u32 \
   match ip dst ${destination} flowid 1:10

tc qdisc add dev ${interface} parent 1:10 handle 20: sfq perturb 10

sleep ${sleep_time}
tc qdisc del dev ${interface} root

exit 0
```

Τέλος για την αύξηση των ενεργών διεργασιών δημιουργήσαμε το παρακάτω πρόγραμμα σε C που δημιουργεί παραμετροποιήσιμο αριθμό διεργασιών για συγκεκριμένο χρονικό διάστημα.

Script 4.5: Spawn Processes

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int main(int argc, char **argv) {
   int i, status;
   pid_t pid;
   int n = 500; //num of children processes
   int sleep_time = 300;

   if (argc == 3) {
      n = atoi(argv[1]);
      sleep_time = atoi(argv[2]);
   }

   for (i=0; i<n; i++) {
      pid = fork();
      if (pid < 0) {
         perror("fork");
         exit(1);
      }
      else if (pid == 0) {
         sleep(sleep_time);
         exit(0);
      }
   }

   while (n > 0) {
      pid = wait(&status);
      n--;
   }

   return 0;
}
```

Επειδή θέλαμε η εισαγωγή ανωμαλιών να γίνει σε συγκεκριμένα χρονικά διαστήματα χωρίς την επέμβασή μας, χρησιμοποιήσαμε και πάλι το εργαλείο cron. Το αρχείο crontab που δημιουργήσαμε είναι το παρακάτω:

Script 4.6: Anomaly Injection Crontab

```
# Run cpu load not during benchmark for 2mins
45 6 7 6 * /usr/bin/stress -c 3 -t 120
# Run cpu load during benchmark for 3mins
11 8 7 6 * /usr/bin/stress -c 2 -t 180
# Run cpu/memory load not during benchmark for 2mins
7 15 7 6 * /usr/bin/stress -m 3 --vm-bytes 512M --vm-keep -t 120
# Run disk load not during benchmark for 5mins
2 3 8 6 * /usr/bin/stress -d 3 --hdd-bytes 1G -t 300
# Run spawn processes not during benchmark for 2 mins
49 6 8 6 * /home/ubuntu/spawn_procs 500 120
# Run spawn processes during benchmark for 2 mins
56 8 8 6 * /home/ubuntu/spawn_procs 334 120
# Run cpu load during benchmark for 40secs
22 9 8 6 * /usr/bin/stress -c 3 -t 40
# Run Network anomaly (wget) not during benchmark
37 14 8 6 * wget -O /dev/null ftp://ftp.ntua.gr/../debian-8.8.0-amd64-DVD-1.iso
# Run disk load during benchmark for 5mins
56 16 8 6 * /usr/bin/stress -d 5 --hdd-bytes 1G -t 300
# Run Network anomaly (wget) during benchmark
28 17 8 6 * wget -O /dev/null ftp://ftp.ntua.gr/../debian-8.8.0-amd64-CD-1.iso
# Run cpu load during benchmark for 40secs
43 1 9 6 * /usr/bin/stress -c 3 -t 40
# Run Network traffic shapping during benchmark for 3mins
48 12 10 6 * /home/ubuntu/traffic_shapping.sh
# Run Network traffic shapping during benchmark for 20mins
37 21 10 6 * /home/ubuntu/traffic_shapping.sh -t 1200
```

4.1.3 Επιλογή Μετρικών

Από τις μετρικές που κατέγραφε το σύστημα παρακολούθησης Ganglia επιλέξαμε να συλλέξουμε τις μετρικής που αναγράφονται στον Πίνακα 4.2. Στις μετρικές αυτές, η συνολική χρήση του επεξεργαστή υπολογίστηκε από το άθροισμα της χρήσης συστήματος (Sustem CPU Usage) και χρήστη (User CPU Usage). Επίσης η χρήση της μνήμης υπολογίστηκε ως η διαφορά της συνολικής μνήμης με την ελεύθερη μνήμη (Free Memory) και τη λανθάνουσα μνήμη (Cached Memory).

Πίνακας 4.2: Μετρικές και Μονάδες Μέτρησης

Μετρική	Μονάδα Μέτρηση
Total CPU Usage	Ποσοστό %
CPU Wait Time	Ποσοστό %
Memory Total	Bytes
Memory Used	Bytes
Disk Total	Bytes
Disk Free	Bytes
Disk IO Time	Δευτερόλεπτα
Network Bytes In	Bytes
Network Bytes Out	Bytes
Total Processes	Φυσικός Αριθμός
Running Processes	Φυσικός Αριθμός

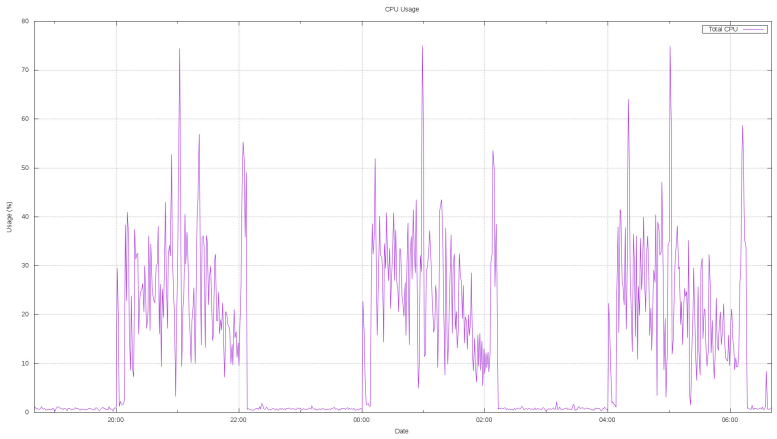
Το Ganglia μας παρέχει κάθε μετρική σε ξεχωριστό RRD αρχείο, για κάθε μηχάνημα που παρακολουθεί. Έτσι για τη συλλογή των μετρικών που μας ενδιέφεραν εξάγαμε τα περιεχόμενα των RRD αρχείων σε μορφή XML και στη συνέχεια έγινε η σύνθεση των μετρικών σε μεμονωμένα αρχεία για κάθε μηχάνημα με το Script Α΄.2. Τα αρχεία που δημιουργούνται διαχωρίζουν τις τιμές με κόμμα (CSV - Comma Separated Values format) και κάθε γραμμή των αρχείων περιλαμβάνει τις τιμές με την σειρά που φαίνονται στο Πίνακα 4.3.

Πίνακας 4.3: Γραμμογράφηση Αρχείων Μετρικών

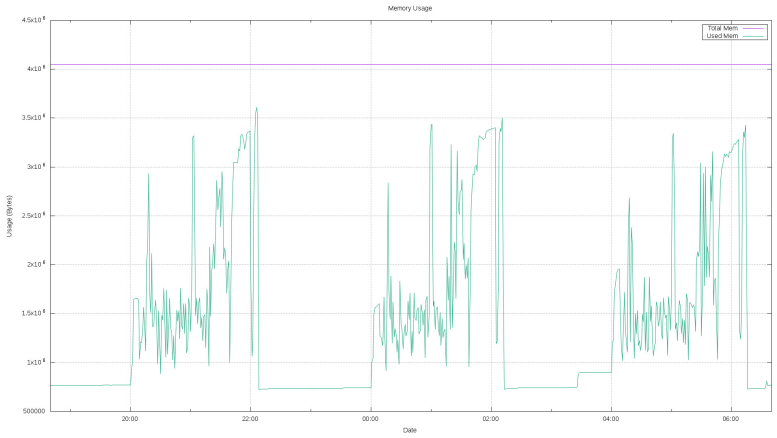
1	Timestamp	7	Disk Free
2	Total CPU Usage	8	Disk IO Time
3	CPU Wait Time	9	Network Bytes In
4	Memory Total	10	Network Bytes Out
5	Memory Used	11	Total Processes
6	Disk Total	12	Running Processes

4.1.4 Περιγραφή Τελικών Αρχείων

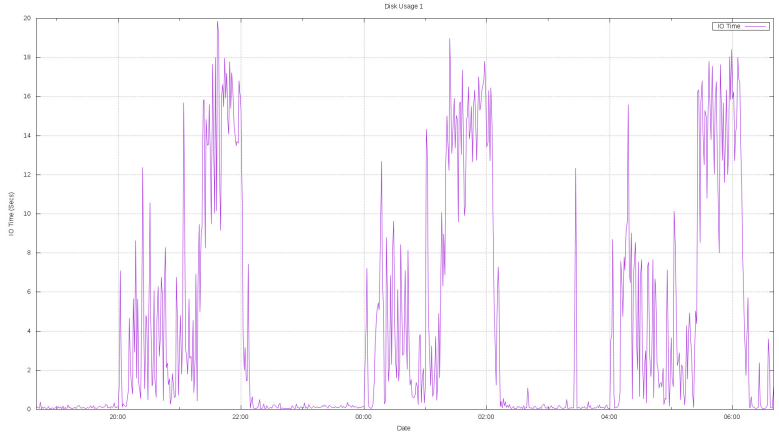
Από την συγχώνευση των μετρικών σε μεμονωμένα αρχεία καταλήγουμε να έχουμε ένα αρχείο για κάθε μηχάνημα (συνολικά 5) που περιέχει τις μετρικές του πίνακα 4.2, με τη γραμμογράφηση που αναφέρεται στον πίνακα 4.3 και βήμα μετρήσεων 10 δευτερολέπτων. Αυτά τα θεωρούμε βασικά αρχεία και παράγουμε αρχεία για 2 ακόμα βήματα μετρήσεων, για 1 λεπτό και για 5 λεπτά. Έτσι καταλήγουμε με συνολικά 15 αρχεία. Ακολουθούν μερικά γραφήματα από τις εκτελέσεις των workloads του HiBench για τον Master κόμβο.



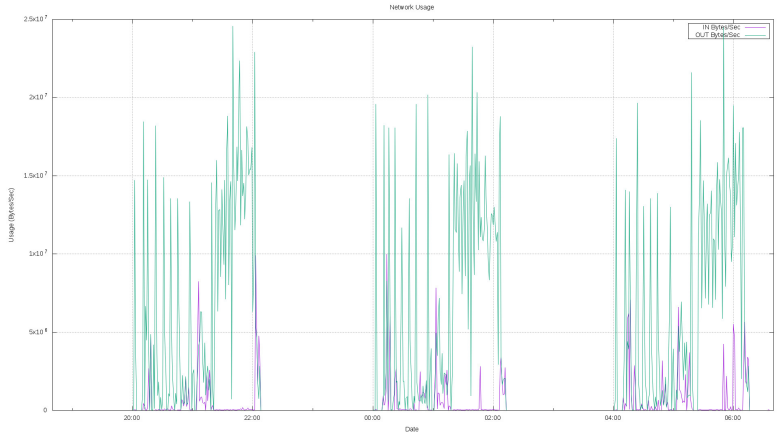
Σχήμα 4.1: Παράδειγμα CPU Usage HiBench



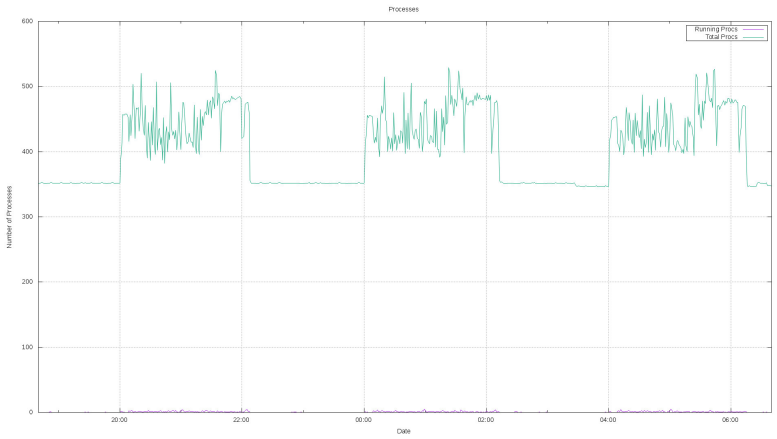
Σχήμα 4.2: Παράδειγμα Memory Usage HiBench



Σχήμα 4.3: Παράδειγμα Disk IO Time HiBench



Σχήμα 4.4: Παράδειγμα Χρήσης Δικτύου HiBench



Σχήμα 4.5: Παράδειγμα Αριθμού Διεργασιών HiBench

4.2 Google Cluster Data

Τα Google Cluster Data [29, 25] είναι ένα σύνολο δεδομένων που έχει δημιουργηθεί από την καταγραφή λειτουργίας μιας συστάδας υπολογιστών (cluster) γενικού σκοπού, πάνω από 12500 κόμβων, σε κάποιο από τα κέντρα δεδομένων (datacenters) της Google. Η πρώτη έκδοση των δεδομένων έγινε διαθέσιμη το 2011 και η τελευταία ανανέωσή τους έγινε το 2014.

4.2.1 Περιγραφή των Δεδομένων

Στα Google Cluster Data βρίσκουμε δεδομένα που έχουν συλλεχθεί σε διάστημα 29 ημερών από ένα Google Compute cell, όπως ονομάζει η Google αυτές τις συστάδες υπολογιστών γενικού σκοπού. Στη διάρκεια των 29 ημερών σε αυτό το Google Compute cell εμφανίζονται 12583 μηχανήματα τα οποία όμως δεν έχουν την ίδια αρχιτεκτονική. Συνολικά τα μηχανήματα κατατάσσονται σε 3 πλατφόρμες των οποίων τα χαρακτηριστικά έχουν αλλοιωθεί για λόγους εμπιστευτικότητας [24].
Η αλλοίωση των δεδομένων για λόγους εμπιστευτικότητας ωστόσο δε σταματάει εδώ, τα ονόματα των χρηστών, των δουλειών και των εργασιών έχουν κατακερματιστεί και η χρησιμοποίηση των πόρων έχει κανονικοποιηθεί στο διάστημα $[0, 1]$ με βάση τη μεγαλύτερη τιμή που παρατηρήθηκε για την κάθε κατηγορία στο σύνολο των δεδομένων.

Το Data Set αυτό μας παρέχει πληροφορίες για τις εξής πτυχές του cluster:

- **Μηχανήματα** (Γεγονότα, Χαρακτηριστικά)
- **Δουλειές και Εργασίες** (Κύκλος ζωής, Γεγονότα)
- **Περιορισμοί Εργασιών**
- **Χρησιμοποίηση Πόρων**

Στη συνέχεια κάνουμε αναφορά στα δεδομένα που παρέχονται για τα γεγονότα μηχανημάτων και για τη χρησιμοποίηση πόρων. Εκτενέστερη ανάλυση των δεδομένων ξεφεύγει από το σκοπό της παρούσας διπλωματικής εργασίας, ωστόσο ο αναγνώστης μπορεί να ανατρέξει στις δημοσιεύσεις "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis" [26] και "Characterizing Machines and Workloads on a Google Cluster" [15] για περισσότερα.

Time and Timestamps

Για την κατανόηση των δεδομένων είναι σημαντικό να αναφερθούμε στο τρόπο καταγραφής του χρόνου.
Κάθε εγγραφή του συνόλου δεδομένων αναπαρίσταται ως ένας 64 bit ακέραιος αριθμός, και δείχνει τον χρόνο σε μικρο-δευτερόλεπτα (microseconds), ξεκινώντας 600 δευτερόλεπτα πριν την αρχή της καταγραφής. Για παράδειγμα ένα γεγονός που συνέβη 20 δευτερόλεπτα μετά την αρχή της καταγραφής θα έχει timestamp ίσο με 620 δευτερόλεπτα.

Επίσης είναι σημαντικό να σημειώσουμε ότι στο Data Set βρίσκουμε δυο χρονικές στιγμές που αναπαριστούν συμβάντα που έγιναν εντός της περιόδου καταγραφής.

- **Χρονική Στιγμή 0**: Αναπαριστά συμβάντα που έγιναν πριν την αρχή της καταγραφής. Όπως για παράδειγμα μηχανήματα που προϋπήρχαν στη συστάδα υπολογιστών, δουλειές και εργασίες που είχαν ήδη υποβληθεί κτλπ.
- **Χρονική Στιγμή $2^{63} - 1$ (MAXINT)**: Αναπαριστά συμβάντα που δεν έγιναν μέσα στο παράθυρο καταγραφής. Όπως για παράδειγμα, μια αστοχία στη συλλογή δεδομένων προς το τέλος του παραθύρου καταγραφής.

Οι χρονικές στιγμές αυτές, όμως, δεν εμφανίζονται στα δεδομένα χρησιμοποίησης πόρων.

Γεγονότα Μηχανημάτων

Κάθε μηχάνημα περιγράφεται από ένα ή περισσότερα γεγονότα στον πίνακα των γεγονότων μηχανημάτων (machine events table). Οι περισσότερες εγγραφές περιγράφουν τα μηχανήματα που βρίσκονται ήδη στο cluster πριν την εκκίνηση της καταγραφής.

Ο πίνακας των γεγονότων για τα μηχανήματα περιλαμβάνει τα εξής πεδία:

Πίνακας 4.4: Google Cluster Data Machine Events Table Row

	Στήλη	Περιγραφή
1	timestamp	Χρονική στιγμή συμβάντος
2	machine ID	64-bit μοναδικό αναγνωριστικό
3	event type	Τύπος γεγονότος
4	platform ID	Το αναγνωριστικό πλατφόρμας
5	capacity-CPU	Κανονικοποιημένη τιμή των διαθ. πυρήνων
6	capacity-Memory	Κανονικοποιημένη τιμή της διαθ. μνήμης

Οι διαθέσιμες τιμές που μπορεί να πάρει ο πεδίο "event type" είναι τρεις:

- **0**: Χαρακτηρίζει την προσθήκη (ADD) κόμβου.
- **1**: Χαρακτηρίζει την αφαίρεση (REMOVE) κόμβου. Αφαίρεση μπορεί να γίνει λόγω αστοχιών ή συντήρησης.
- **2**: Χαρακτηρίζει την ανανέωση (UPDATE) του κόμβου. Η ανανέωση δηλώνει ότι οι πόροι ενός κόμβου άλλαξαν.

Χρησιμοποίηση Πόρων

Η συστάδα υπολογιστών χρησιμοποιεί Linux Containers για την απομόνωση των πόρων και τον υπολογισμό της χρησιμοποίησης τους. Οι μετρήσεις συνήθως γίνονται σε διαστήματα των 300 δευτερολέπτων (5 λεπτών) και αφορούν την εργασία που εκτελείται σε κάποιο container. Ωστόσο μπορούν να υπάρξουν φορές που το διάστημα μέτρησης είναι μικρότερο των 5 λεπτών γιατί η εργασία ενημερώθηκε.
Σε κάθε περίοδο μετρήσεων, λαμβάνονται δείγματα συνήθως κάθε 1 δευτερόλεπτο. Αυτό, όμως, δε συμβαίνει πάντα καθώς ο φόρτος συστήματος το αποτρέπει κάποιες φορές και για αυτό το λόγο σε κάθε εγγραφή δίνεται το πλήθος των δειγμάτων. Τα δείγματα αυτά στη συνέχεια προστίθενται για κάθε περίοδο μέτρησης και βγαίνει ο μέσος όρος χρησιμοποίησης πόρων για την περίοδο.
Επίσης πρέπει να σημειωθεί ότι κάποιες φορές οι μετρήσεις για ένα container μπορεί να προέρχονται από μετρήσεις πολλαπλών sub-containers και σε αυτές τις περιπτώσεις οι μέγιστες τιμές που αναφέρονται είναι αποτέλεσμα της άθροισης των μέγιστων τιμών από τα sub-containers.
Ο πίνακας χρησιμοποίησης πόρων περιλαμβάνει τα εξής πεδία:

Πίνακας 4.5: Google Cluster Data Task Resource Usage Table Row

	Στήλη	Περιγραφή
1	start time	Χρόνος έναρξης μέτρησης
2	end time	Χρόνος τέλους μέτρησης
3	job ID	Αναγνωριστικό δουλειάς στην οποία ανήκει η εργασία
4	task index	Δείκτης εργασίας, σχετικός με τη δουλειά
5	machine ID	64-bit μοναδικό αναγνωριστικό
6	mean cpu	Μέση χρησιμοποίηση του cpu
7	canonical memory	Μνήμη σε χρήση
8	assigned memory	Μνήμη που ανατέθηκε στο container
9	unmapped page cache	Μνήμη cache που δεν ανήκει σε κάποια εφαρμογή στο χώρο χρήστη
10	total page cache	Συνολική μνήμη cache
11	max memory	Μέγιστη χρήση μνήμης
12	mean disk I/O time	Μέσος χρόνος απασχόλησης δίσκου
13	mean local disk space	Μέση χρήση χώρου τοπικού δίσκου
14	max CPU	Μέγιστη χρήση επεξεργαστή
15	max disk IO time	Μέγιστος χρόνος απασχόλησης
16	cycles per instruction	Κύκλοι επεξεργαστή ανα εντολή
17	mem accesses per inst	Προσβάσεις μνήμης ανά επεξεργαστή
18	sample portion	Αριθμός δειγμάτων
19	aggregation type	Υπολογισμός χρήσης πόρων από subcontainers
20	sampled CPU	Χρήση επεξεργαστή σε τυχαίο δευτερόλεπτο της μέτρησης

Το πεδίο "aggregation type" μπορεί να πάρει τις τιμές 0 και 1. Παίρνει τιμή 1 στην περίπτωση που ο υπολογισμός των μέγιστων τιμών προέρχεται από την άθροιση των μέγιστων τιμών των sub-containers και 0 σε όλες τις άλλες περιπτώσεις.

4.2.2 Επιλογή Μετρικών και Μετατροπή Μετρήσεων ανά Μηχάνημα

Για τις δικές μας ανάγκες επεξεργαστήκαμε τα δεδομένα που μας δίνονται στο Data Set για τη χρησιμοποίηση πόρων και τα μετατρέψαμε σε χρησιμοποίηση πόρων ανά μηχάνημα. Για τη μετατροπή αυτή χρησιμοποιήσαμε σταθερά παράθυρα των 5 λεπτών και πάνω σε αυτά ομαδοποιήσαμε τις εργασίες που εκτελούνταν σε κάθε μηχάνημα και εξάγαμε την μέση χρησιμοποίηση συγκεκριμένων πόρων συστήματος.

Από τα πεδία που αναγράφονται στον πίνακα 4.5 επιλέξαμε να συλλέξουμε τα παρακάτω:

- **mean cpu**
- **canonical memory**
- **assigned memory**
- **mean disk I/O time**
- **mean local disk space**

Η επιλογή αυτή έγινε βάσει δύο κριτηρίων. Ποιες μετρικές μπορούμε να ανασυνθέσουμε συνολικά για κάθε μηχάνημα και ποιες μετρικές θα μπορούν να μας δώσουν κάποια χρήσιμη πληροφορία.

Για την μετατροπή αυτή χρησιμοποιήσαμε το script Α′.3.

4.2.3 Περιγραφή Τελικών Αρχείων

Τελικά καταλήγουμε με ένα αρχείο χρησιμοποίησης πόρων για κάθε μηχάνημα και παράγουμε ένα ακόμα αρχείο που περιέχει όλες τις εγγραφές από τα επιμέρους αρχεία. Οι δύο αυτοί τύποι αρχείων, διαθέτουν διαφορετική γραμμογράφηση.
Τη γραμμογράφηση των αρχείων για κάθε μηχάνημα μπορούμε να τη δούμε στον πίνακα 4.6. Ενώ τη γραμμογράφηση του αρχείου που περιέχει όλες τις εγγραφές μπορούμε να τη δούμε στον πίνακα 4.7.

Πίνακας 4.6: Γραμμογράφηση Αρχείων Μετρικών ανά Μηχάνημα (Google Cluster Data)

	Τίτλος	Σχόλια
1	Measurement Start	Δευτερόλεπτα
2	Measurement End	Δευτερόλεπτα
3	Mean CPU Usage	Κανονικοποιημένη Τιμή
4	Mean Cannonical Mem	Κανονικοποιημένη Τιμή
5	Mean Disk I/O Time	Κανονικοποιημένη Τιμή
6	Mean Local Disk Space	Κανονικοποιημένη Τιμή

Πίνακας 4.7: Γραμμογράφηση Αρχείου Μετρικών όλων των Μηχανημάτων (Google Cluster Data)

1	Timestamp (end of measurement)
2	Machine ID
3	Mean CPU Usage
4	Mean Cannonical Mem
5	Mean Disk I/O Time
6	Mean Local Disk Space

Κεφάλαιο 5

Αξιολόγηση Τεχνικής

Σε αυτό το κεφάλαιο παρουσιάζουμε κάποια στατιστικά για την αποδοτικότητα της μεθόδου που χρησιμοποιήσαμε.
Βασιζόμενοι στο σύνολο δεδομένων που δημιουργήσαμε και παρουσιάσαμε στο κεφάλαιο 4 εκτελούμε προσομοιώσεις για να αξιολογήσουμε την τεχνική εύρεσης ανωμαλιών, πάνω σε δεδομένα που εμφανίζουν επαναληπτικότητα.

Για την αποτίμηση χρησιμοποιούμε τους παρακάτω 3 ορισμούς:

- Precision (Ακρίβεια): Δείχνει πόσες πραγματικά ανωμαλίες εντοπίσαμε συνολικά.
  $p = \frac{TruePositive}{TruePositive+FalsePositive}$
- Recall (Ανάκληση): Δείχνει πόσες από τις επιβεβαιωμένες ανωμαλίες εντοπίσαμε.
  $r = \frac{TruePositive}{TruePositive+FalseNegative}$
- False-Out: Δείχνει πόσες λανθασμένες ανωμαλίες εντοπίσαμε.
  $f = \frac{FalsePositive}{TruePositive+FalsePositive}$

5.1 Δοκιμές στο custom σύνολο δεδομένων

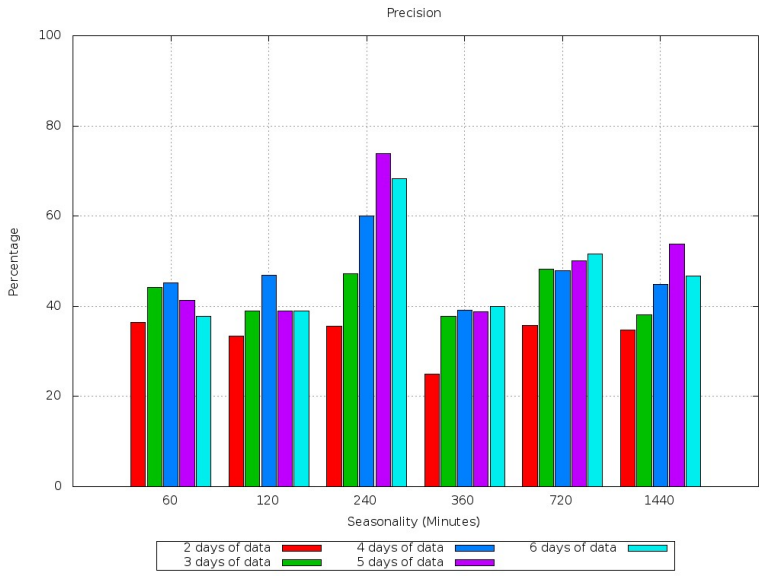
Όπως αναφέραμε στο κεφάλαιο 4, στο σύνολο δεδομένων που δημιουργήσαμε, εισάγαμε ανωμαλίες στον κόμβο "Slave-3". Έτσι παρακάτω ελέγχουμε την τεχνική εύρεσης ανωμαλιών με τα δεδομένα που συλλέχθηκαν για τον κόμβο αυτό.

Ποιο συγκεκριμένα εκτελούμε προσομοιώσεις για διαφορετικό πλήθος ιστορικών δεδομένων και για διαφορική εποχικότητα.

Για τα ιστορικά δεδομένα δοκιμάστηκαν: 2 ημέρες, 3 ημέρες, 4 ημέρες, 5 ημέρες και 6 ημέρες

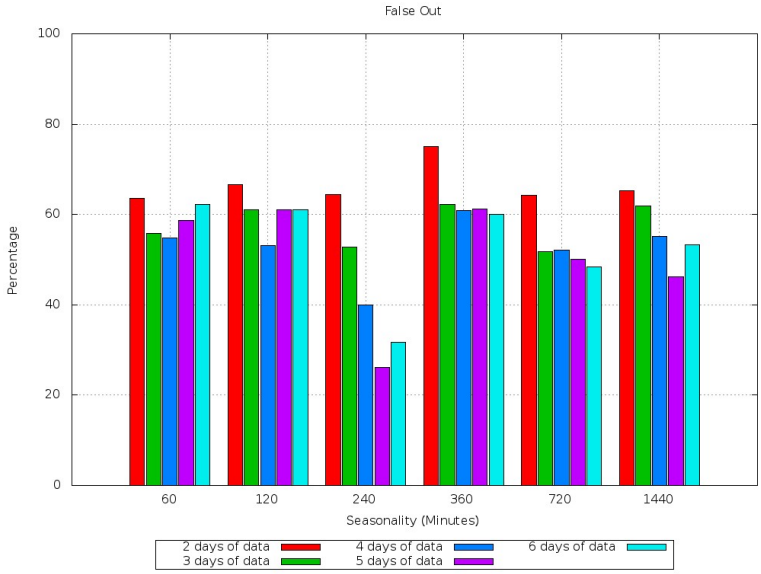
Για την εποχικότητα δοκιμάστηκαν: 60 λεπτά, 120 λεπτά, 240 λεπτά, 360 λεπτά, 720 λεπτά, 1440 λεπτά

Στο σχήμα 5.1 αποτυπώνεται η ακρίβεια του αλγορίθμου για τους διάφορους συνδυασμούς ιστορικών δεδομένων και εποχικότητας. Παρατηρούμε ότι για εποχικότητα ίση με 240 λεπτά και ιστορικά δεδομένα 5 ημερών ο αλγόριθμός παρουσιάζει την καλύτερη ακρίβεια (∼73%), με δεύτερη καλύτερη ακρίβεια για ιστορικά δεδομένα 6 ημερών (∼68%). Αυτό δεν είναι τυχαίο καθώς τα benchmarks από τη σουίτα HiBench εκτελούνταν κάθε 4 ώρες, οπότε μπορούμε να διακρίνουμε μια ισχυρή σχέση μεταξύ της εποχικότητας τον δεδομένων και τις εποχικότητας της ανάλυσης.
Επιπλέον, αν εξαιρέσουμε τις εποχικότητες 60 και 120 λεπτών, παρατηρούμε ότι όσο αυξάνονται τα ιστορικά δεδομένα, αποκτούμε και καλύτερη ακρίβεια. Παρ' όλα αυτά είναι εμφανές ότι για εποχικότητα διαφορετική από αυτή των δεδομένων, η ακρίβεια πέφτει πολύ χαμηλά, αρκετές φορές και κάτω από το 50%.
Τέλος, θα περίμενε κανείς ότι για εποχικότητες 720 και 1440 λεπτών, η ακρίβεια δε θα ήταν κακή, γιατί είναι ακέραια πολλαπλάσια της εποχικότητας των δεδομένων. Αντιθέτως, όμως, η ακρίβεια βρίσκεται κάτω από το 60% και αυτό μπορεί να διακιολογηθεί με τις διαφοροποιήσεις στους χρόνους εκτέλεσης των benchmarks. Λόγω του ότι κάθε φορά διαγράφαμε και εισάγαμε ξανά τα δεδομένα στο hdfs cluster, διατηρώντας ένα αντίγραφο για εξοικονόμηση χώρου, ο φόρτος εκτέλεσης άλλαζε λίγο από κόμβο σε κόμβο με αποτέλεσμα να παράγονται διαφορετικοί χρόνοι εκτέλεσης.



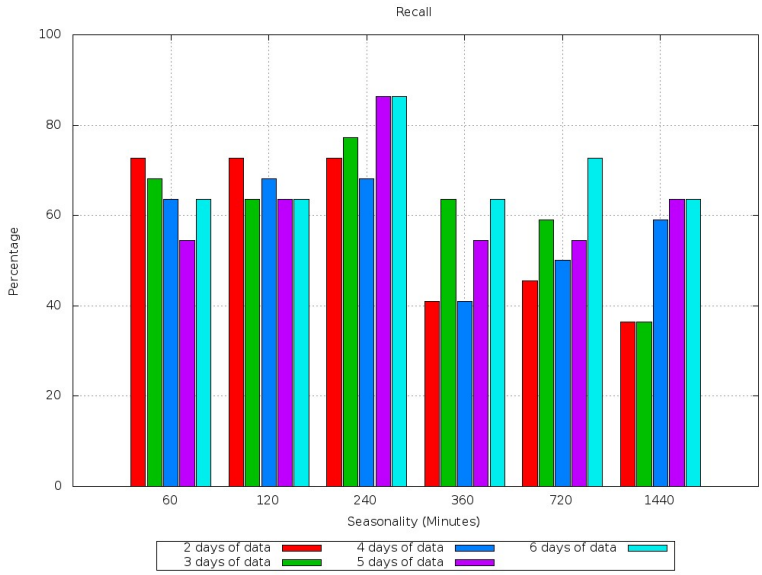
Σχήμα 5.1: Ακρίβεια εντοπισμού με μεταβλητή εποχικότητα και πλήθος ιστορικών σημείων

Στο σχήμα 5.2 παρουσιάζεται το ποσοστό λανθασμένων αναγνωρισμένων ανωμαλιών και αποτελεί ένα συμμετρικό γράφημα ως προς το 5.1.



Σχήμα 5.2: Αστοχίες εντοπισμού με μεταβλητή εποχικότητα και πλήθος ιστορικών σημείων

Στο σχήματα 5.3 παρουσιάζεται το ποσοστό αναγνωρισμένων ανωμαλιών από το σύνολο των ανωμαλιών, για κάθε συνδυασμό πλήθους ιστορικών δεδομένων και εποχικότητας. Και πάλι παρατηρούμε ότι για εποχικότητα ίση με 240 λεπτά λαμβάνουμε την καλύτερη ανάκληση (∼86%) και αυτό συμβαίνει τόσο για 5 όσο και για 6 ημέρες ιστορικών δεδομένων. Παρατηρούμε επίσης ότι η ανάκληση για μικρές εποχικότητες και λίγα ιστορικά δεδομένα, κυμαίνεται σε ποσοστά άνω του 70%, ωστόσο αυτό οφείλεται στον τρόπου υπολογιζόταν το Top X% των ανωμαλιών, αφού εφαρμοζόταν στο σύνολο των δεδομένων, κι όχι απλά αυτών προς έλεγχο.
Αντιθέτως για μεγαλύτερες εποχικότητες από 240 λεπτά, η ανάκληση ξεπερνάει το 70% μόνο για εποχικότητα 720 λεπτών και πλήθους ιστορικών σημείων 6 ημερών. Στις άλλες περιπτώσεις το ποσοστό είναι πολύ χαμηλό και κυμαίνεται απο ∼36% μέχρι ∼63%.
Όπως και για το γράφημα της ακρίβειας, έτσι κι εδώ μπορούμε να αποδώσουμε αυτό το φαινόμενο στο μεταβλητό χρόνο εκτέλεσης, αλλά και στον τρόπο υπολογισμού του Top X% των πιο σημαντικών ανωμαλιών.

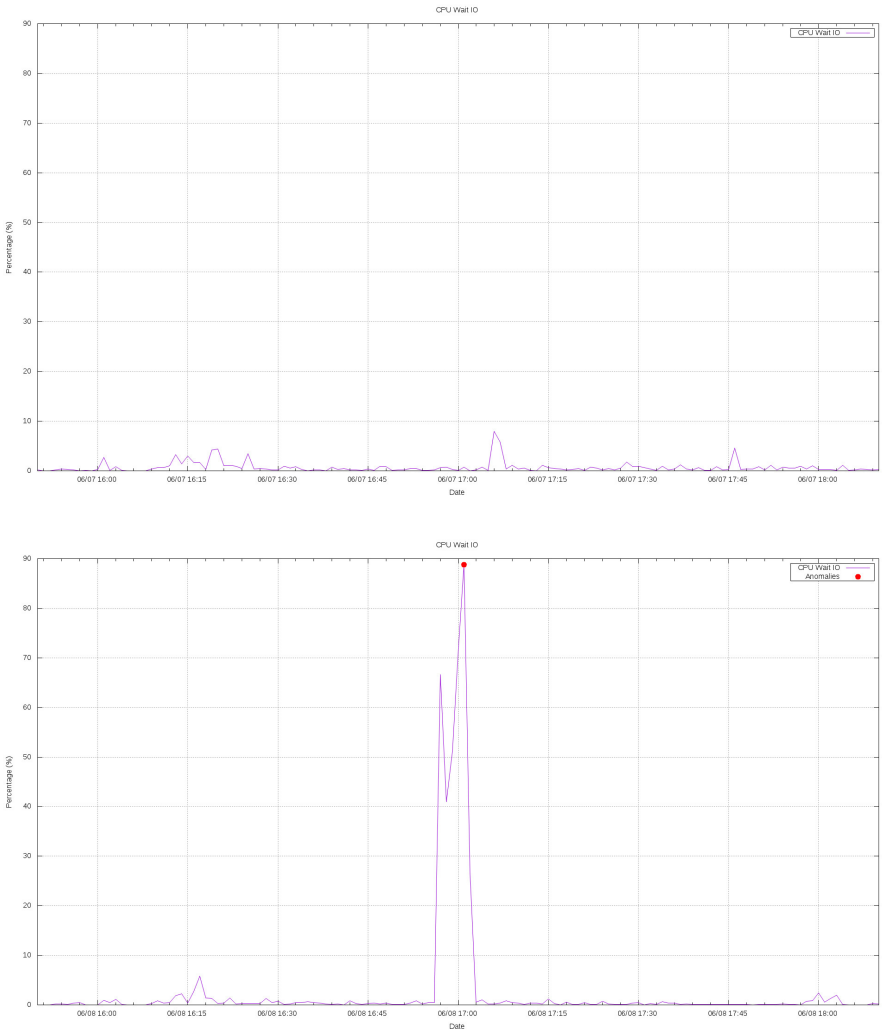


Σχήμα 5.3: Ανάκληση εντοπισμού με μεταβλητή εποχικότητα και πλήθος ιστορικών σημείων

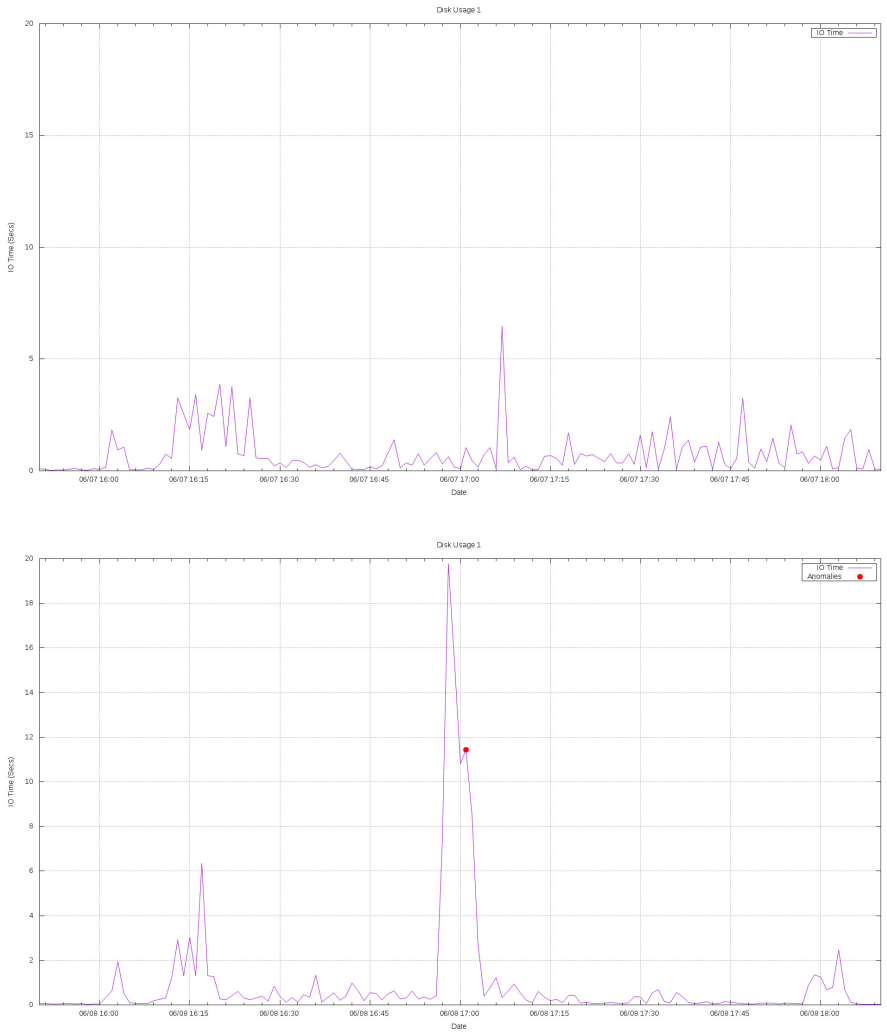
5.1.1 Στιγμιότυπα Εντοπισμένων Ανωμαλιών

Στα σχήματα αυτής της ενότητας παρουσιάζουμε στιγμιότυπα που περιέχουν αναγνωρισμένες ανωμαλίες από τον κόμβο "Slave-3" και τα παραθέτουμε με αντίστοιχα στιγμιότυπα προηγούμενης ημέρας που δεν περιέχουν ανωμαλίες. Για την επίτευξη εμφανής σύγκρισης το κάθε ζευγάρι σχημάτων ακολουθεί ίδια όρια στον κάθετο άξονα.
Στο σχήμα 5.4 παρουσιάζουμε ένα στιγμιότυπο από αυξημένη χρήση επεξεργαστή.
Στο σχήμα 5.5 παρουσιάζουμε ένα στιγμιότυπο από αυξημένη αναμονή IO από τον επεξεργαστή (cpu wait IO).
Στο σχήμα 5.6 παρουσιάζουμε ένα στιγμιότυπο με αυξημένο χρόνο IO στο δίσκο. Σε αυτό το γράφημα αξίζει να σημειώσουμε ότι η αναγνωρισμένη ανωμαλία δε βρίσκεται στο μέγιστο της κατανάλωσης αλλά λίγο πιο μετά.
Στο σχήμα 5.7 παρουσιάζουμε ένα στιγμιότυπο στο οποίο αυξήσαμε το πλήθος των διεργασιών.
Στο σχήμα 5.8 παρουσιάζουμε ένα στιγμιότυπο αυξημένης κίνησης δικτύου.

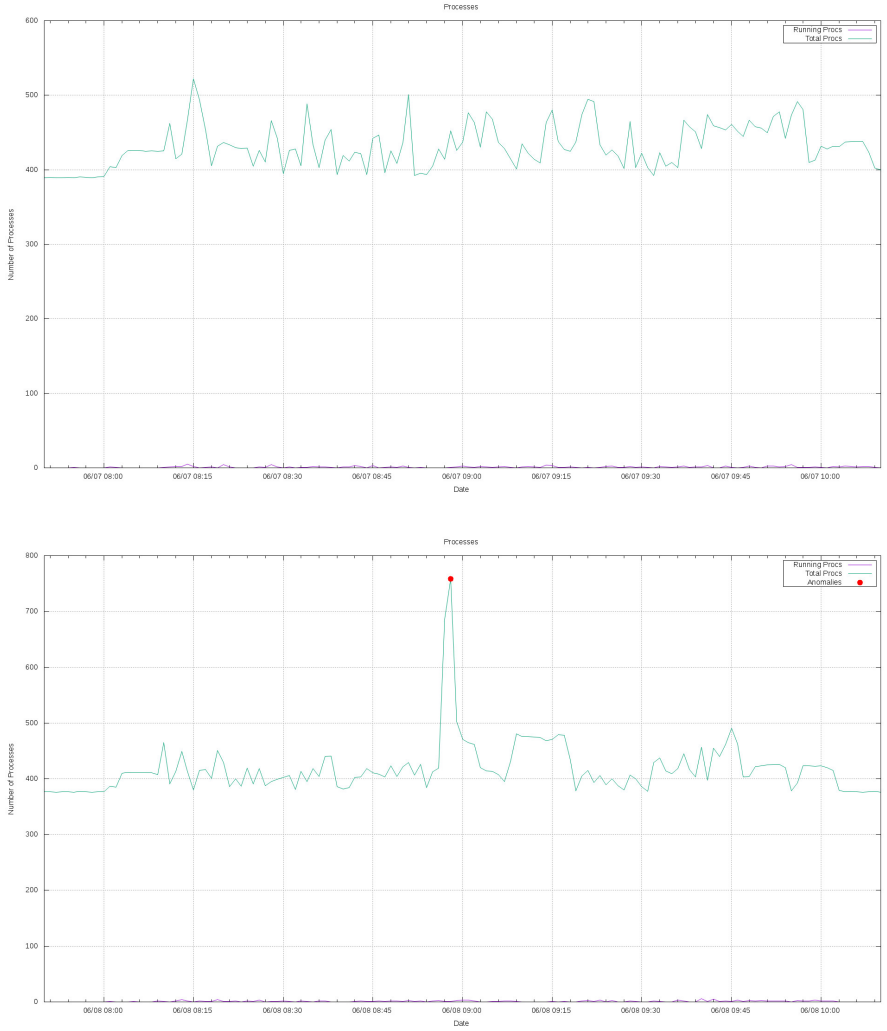
Σχήμα 5.4: Στιγμιότυπο ανωμαλίας CPU



Σχήμα 5.5: Στιγμιότυπο ανωμαλίας CPU Wait IO



Σχήμα 5.6: Στιγμιότυπο ανωμαλίας δίσκου



Σχήμα 5.7: Στιγμιότυπο ανωμαλίας πλήθους διεργασιών

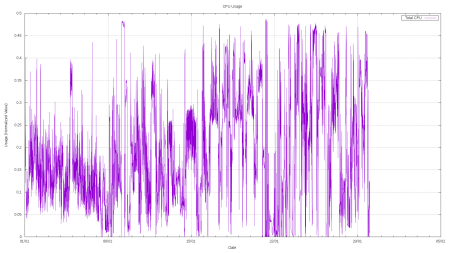
Σχήμα 5.8: Στιγμιότυπο ανωμαλίας δικτύου

5.1.2 Επιπλέον Σχολιασμός

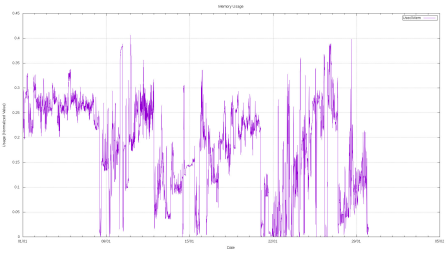
Κατά τις δοκιμές που διεξήχθησαν, είχαμε στη διάθεσή μας 3 σύνολα δεδομένων, που περιέγραφαν τις ίδιες μετρήσεις αλλά με διαφορετικά βήματα. Στο πρώτο σύνολο δεδομένων είχαμε βήμα 10 δευτερολέπτων, στο δεύτερο 60 δευτερολέπτων (1 λεπτό) και στο τρίτο 300 δευτερολέπτων (5 λεπτών).
Δοκιμάζοντας την τεχνική με το βήμα 10 δευτερολέπτων παρατηρήθηκαν πάρα πολλές εσφαλμένες αναγνωρίσεις, κυρίως στην κατανάλωση επεξεργαστή, καθώς παρουσιάζει πολύ απότομες μεταβολές.
Από την άλλη, δοκιμάζοντας την τεχνική με δεδομένα βήματος 5 λεπτών, είχαμε πολύ λίγες αναγνωρίσεις ανωμαλιών και χαμηλή ανάκληση. Αυτό οφείλεται στο γεγονός ότι οι ανωμαλίες που εισάγαμε ήταν μικρής διάρκειας και παίρνοντας των μέσο όρο των τιμών για 5 λεπτά, μείωνε κατά πολύ τη συνεισφορά τους.
Τέλος για 1 λεπτό είχαμε καλύτερα αποτελέσματα, αφού ήταν μια καλή μέση λύση μεταξύ των απότομων μεταβολών και της μείωσης της συνεισφοράς στο μέσο όρο.

5.2 Μελέτη Google Cluster Data

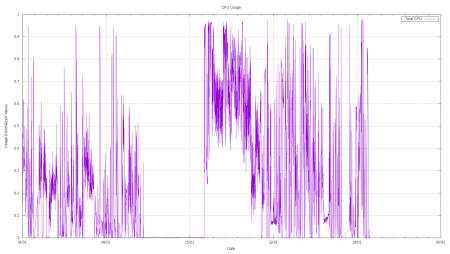
Ανακτώντας και μελετώντας τα αναδομημένα δεδομένα ανά κόμβο των Google Cluster Data, διαπιστώθηκε ότι οι πόροι των υπολογιστικών κόμβων δεν παρουσιάζουν γενικά εποχικότητα. Αυτό μπορούμε να το αποδώσουμε στο ότι αποτελούσαν κόμβους μιας γενικής υπολογιστικής κυψέλης που λάμβανε εργασίες υποβεβλημένες από χρήστες χωρίς κάποιο ιδιαίτερο προγραμματισμό. Στα παρακάτω παραδείγματα παραθέτουμε τις τιμές της μέσης χρήσης CPU και RAM για 2 μηχανήματα που με αναγνωριστικά 4478867098/6201459631, που δεν εμφανίζουν εποχικότητα.



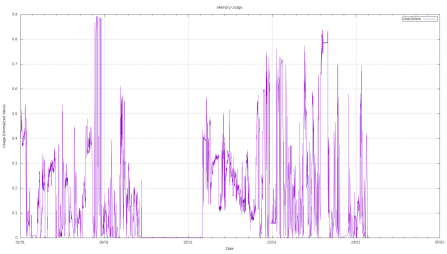
CPU of Machine 4478867098



MEM of Machine 4478867098



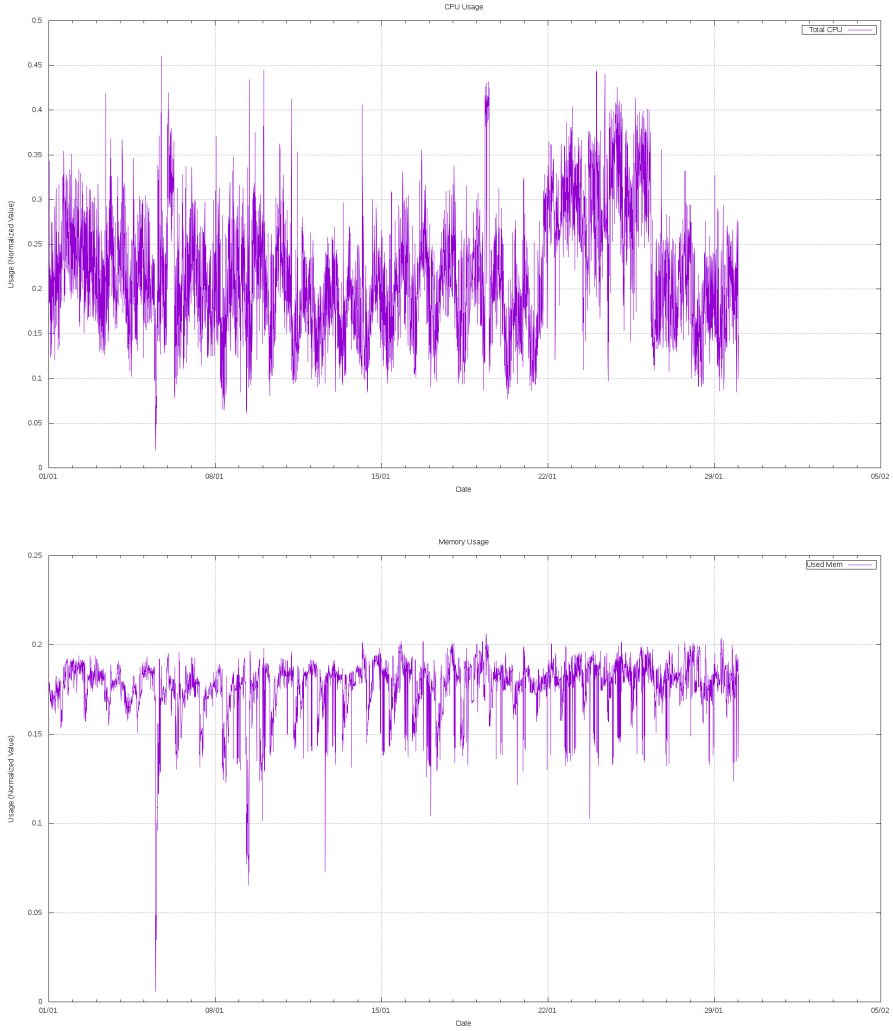
CPU of Machine 6201459631



MEM of Machine 6201459631

Πίνακας 5.1: Στιγμιότυπα κόμβων 4478867098 και 6201459631

Ωστόσο ανάμεσα στους κόμβους υπάρχουν και ορισμένοι που παρουσιάζουν εποχικότητα στην κατανάλωση των πόρων τους, όπως για παράδειγμα ο κόμβος 317497335, όμως δεν αποτελούν τη νόρμα και σίγουρα δεν είναι πρακτικός ο έλεγχος ύπαρξης εποχικότητας για κάθε κόμβο ξεχωριστά όταν αναφερόμαστε σε τόσους πολλούς.



Σχήμα 5.9: Στιγμιότυπο κόμβου 317497335

5.2.1 Επιπλέον Σχολιασμός

Για την περίπτωση δεδομένων όπως των Google Cluster Data, πρέπει να ακολουθηθεί διαφορετική στρατηγική εντοπισμού ανωμαλιών. Μια ιδέα βασισμένη στην τεχνική που ακολουθήθηκε μπορεί να είναι η εξής:

Δεδομένου ότι γνωρίζουμε τα ονόματα εργασιών που υποβάλλονται στους κόμβους είναι πιθανή η εφαρμογή εντοπισμού ανωμαλιών σε επίπεδο εργασίας και αρχιτεκτονικής κόμβου, χρησιμοποιώντας ως ιστορικά δεδομένα προηγούμενες εκτελέσεις της διεργασίας σε κόμβους ίδιας αρχιτεκτονικής.
Ωστόσο αυτή η ιδέα περιέχει αρκετές υποθέσεις, όπως για παράδειγμα ότι όλες οι εργασίες έχουν ίδια δικαιώματα πάνω στους πόρους, οι κόμβοι ίδιας αρχιτεκτονικής διαθέτουν πόρους παρόμοιας απόδοσης κα.

Κεφάλαιο 6

Εκτέλεση με Apache Spark

6.1 Περιγραφή

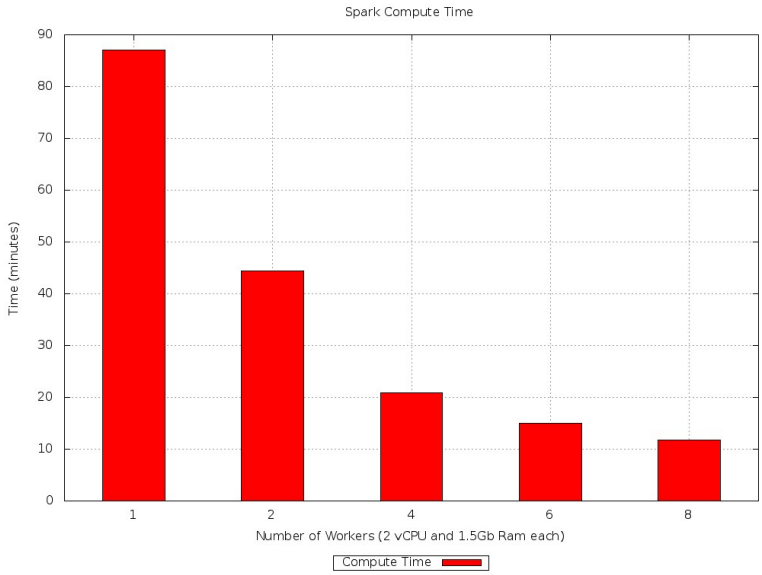
Ο αλγόριθμος εύρεσης ανωμαλιών ανά πόρο συστήματος, όπως παρουσιάστηκε στο κεφάλαιο 3, μπορεί να υλοποιηθεί σε μηχανές κατανεμημένης ανάλυσης όπως το Apache Spark και να επιταχυνθεί η εκτέλεση της ανάλυσης.
Στο κεφάλαιο αυτό υποθέτουμε ότι η ανάλυση αυτή εκτελείτε ημερησίως ή μερικές φορές μέσα στην ημέρα για όλους τους κόμβους που παρακολουθούνται.

6.2 Αξιολόγηση Επίδοσης

Για την αξιολόγηση της κλιμάκωσης, υλοποιήθηκε ένα cluster από 5 κόμβους στην υποδομή Openstack του εργαστηρίου και εγκαταστάθηκαν τα πακέτα Apache Hadoop 2.7.5, Apache HBase 1.2.6 και Apache Spark 2.2.1. Στην HBase δημιουργήθηκε ένας πίνακας με όνομα "resource_usage", και οργανώθηκε σε πολλαπλά column families, ένα για κάθε τύπο πόρου (πχ. Επεξεργαστή, Μνήμη, Δίκτυο κτλπ.) Τέλος τα δεδομένα έγιναν bulk insert από ένα αρχείο TSV στην HBase μέσω του εργαλείου importtsv που διανέμεται μαζί με την βάση.

Τα βήματα που ακολουθούνται για την επίλυση στο Apache Spark είναι τα ακόλουθα:

- Ανάκτηση των δεδομένων για κάθε κόμβο με scan operation από την HBase σε Spark Dataframe
- Μετατροπή του Dataframe σε Resilient Distributed Dataset από Vectors μετρικών
- Κανονικοποίηση με τον Min-Max Scaler της βιβλιοθήκης Spark MLlib
- Εφαρμογή του RobustPCA (A΄.1) με Map operation στο RRD με τα scaled vectors



Σχήμα 6.1: Χρόνος εκτέλεσης σε Apache Spark με διαφορετικό πλήθος εργατών

6.3 Σχόλια

Στο παραπάνω γράφημα δεν συμπεριλαμβάνουμε στο χρόνο εκτέλεσης το χρόνο που χρειάζεται για να φορτωθούν τα δεδομένα, ο χρόνος που απεικονίζεται περιλαμβάνει καθαρά τον χρόνο που σπαταλήθηκε στον υπολογισμό.
Η εκτέλεση στο Apache Spark καθώς αυξάνονται οι εργάτες (workers), εμφανίζει μείωση του χρόνου εκτέλεσης. Μάλιστα στον διπλασιασμό των εργατών ο χρόνος εκτέλεσης υποδιπλασιάζεται. Αυτό είναι λογικό γιατί καθώς η επεξεργασία πάνω σε κάθε Vector του RDD είναι ανεξάρτητη, ο Spark Master εφαρμόζοντας το Map operation, μπορεί να μοιράσει τις εργασίες παράλληλα στους εργάτες.
Σε αυτό το σημείο φαίνεται η δύναμη του Apache Spark σαν μηχανή εκτέλεσης, καθώς από μια σειριακή εκτέλεση μεταβήκαμε σε μια παράλληλη εκτέλεση (coarse grained), απλά χρησιμοποιώντας ορισμένες από τις δομές δεδομένων και τις έτοιμες συναρτήσεις του.

Κεφάλαιο 7

Επίλογος

7.1 Σύνοψη και Σχόλια

Ο μη επιβλεπόμενος εντοπισμός ανωμαλιών με την τενχική που ακολουθήθηκε στην παρούσα διπλωματική δεν μπορεί να θεωρηθεί αποδοτικός και αρκετά ακριβής για να τον εμπιστευτεί κάποιος τυφλά, καθώς αναφέρει ως ανωμαλίες αρκετά λανθασμένα σημεία. Πιθανότατα μπορεί να βελτιωθεί η απόδοσή του, αλλά αυτό αφήνεται ως μελλοντική εργασία. Επιπλέον καθίσταται ακατάλληλη τεχνική για δεδομένα που δεν εμφανίζουν εποχικότητα, όπως αυτά των Google Cluster Data, καθώς η βασική της αρχή βρίσκεται πίσω από αυτή την προϋπόθεση.

Η τεχνική αποδόμησης πινάκων βασισμένη στην μέθοδο εναλλασσόμενης κατεύθυνσης γινομένων (Alternating Direction Method of Multipliers), φαίνεται αρκετά υποσχόμενη για την κοινότητα που ασχολείται με ανάλυση δεδομένων και μηχανική μάθηση, και η δυνατότητα παράλληλης υλοποίησής, ανοίγει το δρόμο για εφαρμογή της σε πίνακες δισεκατομμυρίων στοιχείων.

7.2 Μελλοντικές Εργασίες

Σαν επέκταση της παρούσας διπλωματικής εργασίας θα μπορούσαμε να κινηθούμε στους παρακάτω άξονες:

- Δημιουργία μιας ολοκληρωμένης λύσης για την εισαγωγή μετρήσεων από τα μηχανήματα ή το σύστημα παρακολούθησης σε μια βάση για την επεξεργασία και εύρεση των ανωμαλιών. Πχ. Χρησιμοποιώντας κάποιον δέμονα (daemon) στα μηχανήματα καταγραφής για την εξαγωγή των μετρήσεων από τη βάση του Ganglia και προώθησή του σε ένα σύστημα publish/subscribe όπως το Apache Kafka.
- Βελτίωση του πλήθους των λανθασμένων αναγνωρίσεων ανωμαλιών. (πχ. χρήση προσαρμοζόμενου ορίου με βάσει προηγούμενες εκτελέσεις ή κάποιας άλλης τεχνικής)

- Εξερεύνηση αν κάποια παραλλαγή της αποδόμησης πινάκων με τον Robust PCA μπορεί να χρησιμοποιηθεί για εντοπισμό των ανωμαλιών σε πραγματικό χρόνο
- Μελέτη τεχνικών που μπορούν να αποδώσουν σε σενάρια εντοπισμού ροών δεδομένων
- Μπορεί να χρησιμοποιηθεί κάποιος μη επιβλεπόμενος τρόπος για τον εντοπισμό ανωμαλιών σε μη περιοδικά και πολύ δυναμικά workloads, (πχ. Google Cluster Data);

Appendices

Παράρτημα Α΄

Επιπρόσθετα Scripts

Script Α΄.1: RobustPCA in Scala

```
import breeze.linalg._
import breeze.numerics._
import scala.util.control._

class RobustPCA(val data: DenseMatrix[Double]) {
  // Number of breakdown matrices
  private var n_matrices: Int = 3
  // Penalty parameter
  protected var lambda: Double = 1.0
  // Parameters related to convergence
  private var max_iter: Int = 1000
  private var eps_abs: Double = 1e-6
  private var eps_rel: Double = 1e-6
  protected var rho: Double = 1.0
  private var logs: Boolean = false
  // Data
  private val dim_x = data.rows
  private val dim_y = data.cols
  //private var data = data
  // Main variables
  protected var admm_x1 = DenseMatrix.zeros[Double](data.rows, data.cols)
  protected var admm_x2 = DenseMatrix.zeros[Double](data.rows, data.cols)
  protected var admm_x3 = DenseMatrix.zeros[Double](data.rows, data.cols)
  // Auxiliary variable
  protected var admm_z = DenseMatrix.zeros[Double](data.rows, this.n_matrices*data.cols)
  // Dual variable
  protected var admm_y = DenseMatrix.zeros[Double](data.rows, data.cols)
  // Number of iterations
  private var iter = 0
  // Residuals and tolerance
  private var eps_primal = 0.0;
  private var eps_dual = 0.0;
  private var resid_primal = 0.0;
  private var resid_dual = 0.0;
```

```
//Proximal operator of the l1 norm for Matrix and Vector
private def prox_l1(data: DenseMatrix[Double], lambdat: Double):
DenseMatrix[Double] = {
  for (i <- 0 until data.rows) {
    for (j <- 0 until data.cols) {
      data.update(i, j,
          math.max(0.0, data(i, j) - lambdat) - math.max(0.0, -data(i, j) - lambdat))
    }
  }
  return data
}
private def prox_l1(data: DenseVector[Double], lambdat: Double):
DenseVector[Double] = {
  for (i <- 0 until data.length) {
    data.update(i,
        math.max(0.0, data(i) - lambdat) - math.max(0.0, -data(i) - lambdat))
  }
  return data
}

//Proximal operator of matrix
private def prox_matrix(data: DenseMatrix[Double], lambdat: Double):
DenseMatrix[Double] = {
  val svd.SVD(u,s,v) = svd.reduced(data)
  val pf = diag(prox_l1(s,lambdat))
  return u*pf*v
}

private def avg3(x1: DenseMatrix[Double], x2: DenseMatrix[Double],
x3: DenseMatrix[Double]): DenseMatrix[Double] = {
  val sumx = x1 + x2 + x3
  return sumx / 3.0
}

private def frobenius_norm(data: DenseMatrix[Double]) : Double = {
  math.sqrt(sum(data.map(x => math.pow(x, 2.0))))
}

// Tolerance for primal residual
private def compute_eps_primal(admm_x123: DenseMatrix[Double]): Double = {
  val t_norm = math.max(frobenius_norm(admm_x123), frobenius_norm(-admm_z))
  return math.sqrt(dim_x * dim_y * n_matrices) * eps_abs + eps_rel * t_norm
}
```

```
// Tolerance for dual residual
private def compute_eps_dual(): Double = {
  val t_norm = math.sqrt(n_matrices) * frobenius_norm(rho * admm_y)
  return math.sqrt(dim_x * dim_y * n_matrices) * eps_abs + eps_rel * t_norm
}

// Dual residual
private def compute_resid_dual(new_z: DenseMatrix[Double]): Double = {
  return frobenius_norm(-rho * (new_z - admm_z))
}

// Primal residual
private def compute_resid_primal (admm_x123: DenseMatrix[Double]): Double = {
  return frobenius_norm(admm_x123 - admm_z)
}

def run() {
  val g2_max = max(data)
  val g3_max = frobenius_norm(data)
  val g2 = 0.15*g2_max
  val g3 = 0.15*g3_max

  val loop = new Breaks
  loop.breakable {
    for(i <- 0 until max_iter) {
      iter = i

      // y step update
      admm_y :+= avg3(admm_x1, admm_x2, admm_x3) - (data / (1.0 * n_matrices))

      // x step update
      admm_x1 = (1.0/(1.0 + lambda)) * (admm_x1 - admm_y)
      admm_x2 = prox_l1(admm_x2 - admm_y, lambda*g2)
      admm_x3 = prox_matrix(admm_x3 - admm_y, lambda*g3)

      // z step update
      val admm_x123 = DenseMatrix.horzcat(admm_x1, admm_x2, admm_x3)
      val new_z = admm_x123 + tile(-avg3(admm_x1, admm_x2, admm_x3)
                + (data / (1.0 * n_matrices)), 1, n_matrices)
      resid_dual = compute_resid_dual(new_z)
      admm_z = new_z

      resid_primal = compute_resid_primal(admm_x123)
```

```
        // Calculate tolerance values
        eps_primal = compute_eps_primal(admm_x123)
        eps_dual = compute_eps_dual()

        // Convergence test
        if(resid_primal < eps_primal && resid_dual < eps_dual) {
          loop.break
        }
      }
    }
  }

  def low_rank : DenseMatrix[Double] = admm_x1.copy
  def sparse_component : DenseMatrix[Double] = admm_x2.copy
  def noise : DenseMatrix[Double] = admm_x3.copy
}
```

Script Α΄.2: Combine HiBench Data to Single File

```
import xml.etree.ElementTree as ET
import datetime as DT
import math, os, argparse

rra_order = {'day': 0, '2-weeks': 1, 'year': 2}
output_file = "combined_ganglia_data.out"

filenames = ['cpu_system', 'cpu_user', 'cpu_wio', 'mem_total',
    'mem_free', 'mem_cached', 'disk_total', 'disk_free',
    'diskstat_vda1_io_time', 'bytes_in', 'bytes_out',
    'proc_run', 'proc_total']

output_headers = ['timestamp', 'cpu_total', 'cpu_wio', 'mem_total',
    'mem_used', 'disk_total', 'disk_free', 'diskstat_vda1_io_time',
    'bytes_in', 'bytes_out', 'proc_run', 'proc_total']

#{step, pdp_per_row, lastrecdate, values[]}
rra_data = {}

def combine_metrics(input_dir, output_dir, rra_selected):
    for f in filenames:
        fpath = os.path.join(input_dir, f+'.xml')

        tree = ET.parse(fpath)
        root = tree.getroot()
        step = int(root.find('step').text)
        lastupdate = root.find('lastupdate').text
        lastrecdate = (int(lastupdate) / 10) * 10

        rras = root.findall('rra')
        rra = rras[rra_selected]
        database = rra.find('database')
        pdp_per_row = int(rra.find('pdp_per_row').text)

        rra_data[f] = {'step': step, 'pdp_per_row': pdp_per_row,
                       'lastrecdate': lastrecdate, 'values': []}
        for row in database.iter('row'):
            value = float(row.find('v').text)
            rra_data[f]['values'].append(value)
```

```
step_list = [rra_data[key]['step'] for key in filenames]
pdp_per_row_list = [rra_data[key]['pdp_per_row'] for key in filenames]
lastrecdate_list = [rra_data[key]['lastrecdate'] for key in filenames]
values_len_list = [len(rra_data[key]['values']) for key in filenames]

step = step_list[0]
pdp_per_row = pdp_per_row_list[0]
num_of_records = values_len_list[0]
recstep = step*pdp_per_row
endrecdate = min(lastrecdate_list)
startrecdate = max(lastrecdate_list) - recstep*(num_of_records-1)

#resize all value lists
for f in filenames:
    stream_end = rra_data[f]['lastrecdate']
    stream_start = stream_end - recstep*(num_of_records-1)
    if stream_end > endrecdate:
        diff = (stream_end - endrecdate)/recstep
        rra_data[f]['values'] = rra_data[f]['values'][:-diff]
        rra_data[f]['lastrecdate'] = endrecdate

    if stream_start < startrecdate:
        diff = (startrecdate - stream_start)/recstep
        rra_data[f]['values'] = rra_data[f]['values'][diff:]

values_len_list = [len(rra_data[key]['values']) for key in filenames]
if not all(x==values_len_list[0] for x in values_len_list):
    print "Something went wrong while resizing values list"
    exit()

baserecdate = startrecdate
out_path = os.path.join(output_dir, output_file)
g = open(out_path, "w+")
for i in xrange(values_len_list[0]):
    cpu_total = (rra_data['cpu_user']['values'][i]
                 + rra_data['cpu_system']['values'][i])
    mem_used = (rra_data['mem_total']['values'][i]
                - rra_data['mem_free']['values'][i]
                - rra_data['mem_cached']['values'][i])
```

```
        output = []
        for key in output_headers:
            if key == 'timestamp':
                output.append(baserecdate)
            elif key == 'cpu_total':
                output.append(cpu_total)
            elif key == 'mem_used':
                output.append(mem_used)
            elif key in ['cpu_system','cpu_user','mem_free','mem_cached']:
                continue
            else: output.append(rra_data[key]['values'][i])
        line = ",".join(map(str, output))
        g.write(line+"\n")
        baserecdate += recstep
    g.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Combine Ganglia Data")
    parser.add_argument('input_dir', metavar='input_dir', type=str,
        nargs=1, help='Input Directory')
    parser.add_argument('output_dir', metavar='output_dir', type=str,
        nargs=1, help='Output Directory')
    parser.add_argument('rra_selected', metavar='rra', type=str, nargs=1,
        help='RRA to collect', choices=['day', '2-weeks', 'year'])
    args = parser.parse_args()
    input_dir = args.input_dir[0]
    output_dir = args.output_dir[0]
    rra_selected = rra_order[args.rra_selected[0]]
    combine_metrics(input_dir, output_dir, rra_selected)
    print("Exiting...")
```

Script Α΄.3: Reconstruct Google Cluster Data Resource Usage per Machine

```
import os, gc, math
import pandas as pd
from collections import defaultdict

root_dir = '/media/sf_clusterdata-2011-2'
output_dir = 'google_data/vm_usage'
output_header = ['sample_start_time','sample_end_time', 'number_of_tasks',
    'extended_task_usage_list', 'mean_cpu', 'canonical_memory',
    'assigned_memory', 'mean_disk_io_time', 'mean_local_disk_space']


#####################################
#### Calculate task usage weight
#####################################
def calculate_task_usage_weight(sample_start_time, sample_end_time,
                      task_measurement_start, task_measurement_end):
    weight = 0.0
    sample_duration = sample_end_time - sample_start_time

    if sample_start_time == task_measurement_start
      and sample_end_time <= task_measurement_end:
        weight = 1.0
    elif sample_start_time == task_measurement_start
      and sample_end_time > task_measurement_end:
        weight = ((task_measurement_end - task_measurement_start)
                 /(sample_duration * 1.0))
    elif sample_start_time < task_measurement_start
      and sample_end_time <= task_measurement_end:
        weight = ((sample_end_time - task_measurement_start)
                 /(sample_duration * 1.0))
    elif sample_start_time < task_measurement_start
      and sample_end_time > task_measurement_end:
        weight = ((task_measurement_end - task_measurement_start)
                 /(sample_duration * 1.0))
    return weight


#####################################
#### Find next machine state
#####################################
```

```
def find_current_machine_state(machine_id, sample_start_time,
                               machine_events, current_machine_state):
    ret_val = False
    #machine events item order [0: 'timestamp', 1: 'event_type',
    #                           2: 'platform_id', 3: 'cpu', 4: 'memory']
    event_list = machine_events[machine_id]
    start_index = current_machine_state[machine_id]['ref']

    for i in xrange(start_index+1, len(event_list)-1):
        if (event_list[i][0] <= sample_start_time \
            or (event_list[i][0] >= sample_start_time \
            and event_list[i][0] <= sample_start_time + sample_duration))\
            and not event_list[i+1][0] - event_list[i][0] < 100000:
            ret_val = True
            current_machine_state[machine_id] = {
                'start': event_list[i][0],
                'update_time': event_list[i+1][0], 'ref': i,
                'cpu': event_list[i][3], 'memory': event_list[i][4]
            }
            break

    if not ret_val and event_list[-1][1] in [0,2] and \
        (event_list[-1][0] <= sample_start_time \
        or (event_list[-1][0] >= sample_start_time \
        and event_list[-1][0] <= sample_start_time + sample_duration)):
        ret_val = True
        current_machine_state[machine_id] = {
            'start': event_list[-1][0], 'update_time': None, 'ref': -1,
            'cpu': event_list[-1][3], 'memory': event_list[-1][4]
        }

    return ret_val

#####################################
#### Output sampled data to file
#####################################
def write_sample_in_file(output_dir, machine_id, record):
    output = []
    for key in output_header:
        if key == 'extended_task_usage_list': output.append(len(record[key]))
        else: output.append(record[key])
```

```
    line = ','.join(map(str, output))
    vm_usage_output =
        open(os.path.join(output_dir, str.format("{0}.dat", machine_id)), 'a+')
    vm_usage_output.write(line+"\n")
    vm_usage_output.close()

####################################
#### MAIN
####################################

machine_events_colnames = [
    'timestamp',
    'machine_id',
    'event_type',
    'platform_id',
    'cpu',
    'memory'
]

task_usage_colnames = [
    'start_time',
    'end_time',
    'job_id',
    'task_index',
    'machine_id',
    'mean_cpu_usage',
    'canonical_memory_usage',
    'assigned_memory_usage',
    'unmapped_page_cache_memory_usage',
    'total_page_cache_memory_usage',
    'maximum_memory_usage',
    'mean_disk_io_time',
    'mean_local_disk_space_used',
    'max_cpu_usage',
    'max_disk_io_time',
    'cpi',
    'mai',
    'sample_portion',
    'aggregation_type',
    'sampled_cpu_usage',
    'measurement_period'
]
```

```
#read machines and events
current_machine_state = {}
machine_events = defaultdict(list)
machine_events_dir = os.path.join(root_dir, 'machine_events')

for fname in sorted(os.listdir(machine_events_dir)):
    if not fname.endswith('.gz'):
        continue
    fpath = os.path.join(machine_events_dir, fname)
    machine_events_df = pd.read_csv(fpath, header=None, index_col=False,
                            compression='gzip', names=machine_events_colnames)

    #itertuples order [1: 'timestamp', 2: 'machine_id', 3: 'event_type',
    #                  4: 'platform_id', 5: 'cpu', 6: 'memory']
    for event in machine_events_df.itertuples():
        #machine events item order
        #[0: 'timestamp', 1: 'event_type', 2: 'platform_id', 3: 'cpu', 4: 'memory']
        machine_events[event[2]].append([event[1], event[3], event[4],
                                         float(event[5]), float(event[6])])
        #current machine state
        #[start, update_time, ref to machine_events list order, cpu, memory]
        if not event[2] in current_machine_state:
            current_machine_state[event[2]] = {
                'start': event[1],
                'update_time': None,
                'ref': 0, 'cpu': float(event[5]),
                'memory': float(event[6])}
        elif math.isnan(current_machine_state[event[2]]['cpu']) or \
             math.isnan(current_machine_state[event[2]]['memory']) or \
             (event[1] - current_machine_state[event[2]]['start']) < 100000:
            current_machine_state[event[2]] = {
                'start': event[1],
                'update_time': None,
                'ref': current_machine_state[event[2]]['ref'] + 1,
                'cpu': float(event[5]),
                'memory': float(event[6])}
        elif current_machine_state[event[2]]['update_time'] is None:
            current_machine_state[event[2]]['update_time'] = event[1]
    # print filename in the end of processing
    machine_events_df = None
    gc.collect()
    print fname
```

```
#process task usage
machine_sampled_usage = {}
sample_duration = 300000000
curr_sample_start_time = 600000000
curr_sample_end_time = curr_sample_start_time + sample_duration
task_usage_dir = os.path.join(root_dir, 'task_usage')

for fname in sorted(os.listdir(task_usage_dir)):
    if not fname.endswith('.gz'):
        continue

    fpath = os.path.join(task_usage_dir, fname)
    task_usage_df = pd.read_csv(fpath, header=None, index_col=False,
                      compression='gzip', names=task_usage_colnames)

    #itertuples order
    #[1: start_time, 2: end_time, 3: job_id, 4: task_index,
    #5: machine_id, 6: mean_cpu_usage, 7: canonical_memory_usage,
    #8: assigned_memory_usage, 9: unmapped_page_cache_memory_usage,
    #10: total_page_cache_memory_usage, 11: maximum_memory_usage,
    #12: mean_disk_io_time, 13: mean_local_disk_space_used,
    #14: max_cpu_usage, 15: max_disk_io_time, 16: cpi, 17: mai,
    #18: sample_portion, 19: aggregation_type, 20: sampled_cpu_usage]
    for task_usage in task_usage_df.itertuples():
        starttime = float(task_usage[1])
        endtime = float(task_usage[2])

        #change sample period
        if curr_sample_end_time <= starttime:
            sample_adjust_factor =
                int(starttime - curr_sample_end_time)/sample_duration + 1
            if sample_adjust_factor > 1:
                print "EMERGENCY: SAMPLE ADJUST FACTOR IS GREATER THAN 1"
            curr_sample_start_time =
                curr_sample_end_time + (sample_adjust_factor - 1) * sample_duration
            curr_sample_end_time += sample_adjust_factor * sample_duration

        #calculate weight of mean measurements
        weight = calculate_task_usage_weight(curr_sample_start_time,
                          curr_sample_end_time, starttime, endtime)
```

```
        #add code for total cluster utiliation
        if not task_usage[5] in machine_sampled_usage:
            machine_sampled_usage[task_usage[5]] = {
                'sample_start_time': curr_sample_start_time,
                'sample_end_time': curr_sample_end_time,
                'number_of_tasks': 1,
                'mean_cpu': float(task_usage[6]) * weight,
                'canonical_memory': float(task_usage[7]) * weight,
                'assigned_memory': float(task_usage[8]) * weight,
                'unmapped_page_cache_memory': float(task_usage[9]) * weight,
                'total_page_cache_memory': float(task_usage[10]) * weight,
                'mean_disk_io_time': float(task_usage[12]) * weight,
                'mean_local_disk_space': float(task_usage[13]) * weight,
                'extended_task_usage_list': []
            }
        elif machine_sampled_usage[task_usage[5]]['sample_start_time'] == \
             curr_sample_start_time:
                machine_sampled_usage[task_usage[5]]['number_of_tasks'] += 1
                machine_sampled_usage[task_usage[5]]['mean_cpu'] +=
                    float(task_usage[6]) * weight
                machine_sampled_usage[task_usage[5]]['canonical_memory'] +=
                    float(task_usage[7]) * weight
                machine_sampled_usage[task_usage[5]]['assigned_memory'] +=
                    float(task_usage[8]) * weight
                machine_sampled_usage[task_usage[5]]['unmapped_page_cache_memory'] +=
                    float(task_usage[9]) * weight
                machine_sampled_usage[task_usage[5]]['total_page_cache_memory'] +=
                    float(task_usage[10]) * weight
                machine_sampled_usage[task_usage[5]]['mean_disk_io_time'] +=
                    float(task_usage[12]) * weight
                machine_sampled_usage[task_usage[5]]['mean_local_disk_space'] +=
                    float(task_usage[13]) * weight
        else:
            #check if current machine status is valid for the old sample range
            #and find a valid status if it isn't
            if not current_machine_state[task_usage[5]]['update_time'] is None and \
               current_machine_state[task_usage[5]]['update_time'] <= \
               machine_sampled_usage[task_usage[5]]['sample_start_time']:
                if not find_current_machine_state(task_usage[5],
                   machine_sampled_usage[task_usage[5]]['sample_start_time'],
                   machine_events, current_machine_state):
                    print "COULD NOT FIND CURRENT MACHINE STATE "+str(task_usage[5])
                    print machine_sampled_usage[task_usage[5]]
                    print machine_events[task_usage[5]]
                    print current_machine_state[task_usage[5]]
```

```
            #write record to file with cpu and memory percentages
            #(lookup the machine capacity)
            write_sample_in_file(output_dir, task_usage[5],
                      machine_sampled_usage[task_usage[5]])

            #keep extended task list in temporary var
            #maybe check if tasks in list don't contribute in this sample period,
            #so previous periods should be calculated
            machine_sampled_usage[task_usage[5]] = {
                'sample_start_time': curr_sample_start_time,
                'sample_end_time': curr_sample_end_time,
                'number_of_tasks': 1,
                'mean_cpu': float(task_usage[6]) * weight,
                'canonical_memory': float(task_usage[7]) * weight,
                'assigned_memory': float(task_usage[8]) * weight,
                'unmapped_page_cache_memory': float(task_usage[9]) * weight,
                'total_page_cache_memory': float(task_usage[10]) * weight,
                'mean_disk_io_time': float(task_usage[12]) * weight,
                'mean_local_disk_space': float(task_usage[13]) * weight,
                'extended_task_usage_list': []
            }

        if curr_sample_end_time < endtime:
            s = machine_sampled_usage[task_usage[5]]['extended_task_usage_list']
            s.append(task_usage)

    # print filename in the end of processing
    task_usage_df = None
    gc.collect()
    print fname

for machine_id in sorted(machine_sampled_usage.keys()):
    write_sample_in_file(output_dir, machine_id, machine_sampled_usage[machine_id])
```

Bibliography

[1] Atef Abdelkefi et al. "Robust Traffic Anomaly Detection with Principal Component Pursuit". In: *Proceedings of the ACM CoNEXT Student Workshop*. CoNEXT '10 Student Workshop. Philadelphia, Pennsylvania: ACM, 2010, 10:1–10:2. URL: http://doi.acm.org/10.1145/1921206.1921217.

[2] *Apache Hadoop*. URL: http://hadoop.apache.org/ (visited on 07/22/2017).

[3] *Apache HBase*. URL: https://hbase.apache.org/ (visited on 07/22/2017).

[4] *Apache Spark*. URL: https://spark.apache.org/ (visited on 07/22/2017).

[5] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[6] Stephen Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Found. Trends Mach. Learn.* 3.1 (Jan. 2011), pp. 1–122. URL: http://dx.doi.org/10.1561/2200000016.

[7] Emmanuel J. Candès et al. "Robust Principal Component Analysis?" In: *J. ACM* 58.3 (June 2011), 11:1–11:37. URL: http://doi.acm.org/10.1145/1970392.1970395.

[8] Robert Cleveland et al. *STL: Seasonal Trend Decomposition Procedure Based on Loess*. 1990. URL: https://www.wessa.net/download/stl.pdf (visited on 07/22/2017).

[9] Apache Software Foundation. *Hive-396: Hive performance benchmarks*. 2009. URL: https://issues.apache.org/jira/browse/HIVE-396 (visited on 07/22/2017).

[10] *HiBench*. URL: https://github.com/intel-hadoop/HiBench (visited on 07/22/2017).

[11] Docker Inc. *Docker*. 2013. URL: https://www.docker.com/ (visited on 07/22/2017).

[12] Y. Jin et al. "Anomaly detection in time series via robust PCA". In: *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. Sept. 2017, pp. 352–355.

[13] Zhouchen Lin et al. "Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix". In: *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)* 61.6 (2009).

[14] *Linux TC*. URL: http://lartc.org/manpages/tc.txt (visited on 07/22/2017).

[15] Z. Liu and S. Cho. "Characterizing Machines and Workloads on a Google Cluster". In: *2012 41st International Conference on Parallel Processing Workshops*. Sept. 2012, pp. 397–403.

[16] Wei-Yin Loh. *Classification and regression trees*. 2011. URL: http://www.stat.wisc.edu/~loh/treeprogs/guide/wires11.pdf (visited on 07/22/2017).

[17] Shiqian Ma Necdet Serhat Aybat Donald Goldfarb. *Efficient Algorithms for Robust and Stable Principal Component Pursuit Problems*. 2013. URL: https://arxiv.org/abs/1309.6976 (visited on 07/22/2017).

[18] Neal Parikh and Stephen Boyd. *Matrix Decomposition via ADMM*. 2014. URL: http://web.stanford.edu/~boyd/papers/prox_algs/matrix_decomp.html (visited on 07/22/2017).

[19] Neal Parikh and Stephen Boyd. *Proximal Algorithms*. 2014. URL: http://web.stanford.edu/~boyd/papers/pdf/prox_slides.pdf (visited on 07/22/2017).

[20] Neal Parikh and Stephen Boyd. "Proximal Algorithms". In: *Found. Trends Optim.* 1.3 (Jan. 2014), pp. 127–239. URL: http://dx.doi.org/10.1561/2400000003.

[21] Andrew Pavlo et al. "A Comparison of Approaches to Large-scale Data Analysis". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD '09. Providence, Rhode Island, USA: ACM, 2009, pp. 165–178. URL: http://doi.acm.org/10.1145/1559845.1559865.

[22] *PCA*. URL: https://en.wikipedia.org/wiki/Principal_component_analysis (visited on 07/22/2017).

[23] *RAD - Outlier Detection On Big Data*. URL: https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d6b0494371cc (visited on 07/22/2017).

[24] C. Reiss, J. Wilkes, and J. L. Hellerstein. "Obfuscatory obscanturism: Making workload traces of commercially-sensitive systems safe to release". In: *2012 IEEE Network Operations and Management Symposium*. Apr. 2012, pp. 1279–1286.

[25] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. *Google cluster-usage traces: format + schema*. Technical Report. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data. Mountain View, CA, USA: Google Inc., Nov. 2011.

[26] Charles Reiss et al. "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis". In: *Proceedings of the Third ACM Symposium on Cloud Computing*. SoCC '12. San Jose, California: ACM, 2012, 7:1–7:13. URL: http://doi.acm.org/10.1145/2391229.2391236.

[27] *Surus*. URL: https://github.com/Netflix/Surus (visited on 07/22/2017).

[28] Carson Wang. *HiBench*. 2016. URL: http://schd.ws/hosted_files/apachebigdata2016/3a/HiBench%20-%20The%20Benchmark%20Suite%20for%20Hadoop%2C%20Spark%20and%20Streaming.pdf (visited on 07/22/2017).

[29] John Wilkes. *More Google cluster data*. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Nov. 2011.

[30] Xiaoming Yuan and Junfeng Yang. "Sparse and low-rank matrix decomposition via alternating direction methods". In: *preprint* 12 (2009).