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

Υπηρεσιοστρεφής πλατφόρμα συλλογής και ανάλυσης δεδομένων μεγάλης κλίμακος σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Δαλκαφούκης

Επιβλέπων : Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2016

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

Υπηρεσιοστρεφής πλατφόρμα συλλογής και ανάλυσης δεδομένων μεγάλης κλίμακος σε περιβάλλοντα υπολογιστικών νεφών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Δαλκαφούκης

Επιβλέπων : Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Ιουνίου 2016.

……...	…….	……..
Ιάκωβος Στ. Βενιέρης	Δήμητρα Θεοδώρα Κακλαμάνη	Νικόλαος Ουζούνογλου
Καθηγητής Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2016

……..
Κωνσταντίνος Δαλκαφούκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στην εποχή των Big Data, τα υπολογιστικά νέφη είναι ιδιαίτερα δημοφιλή καθώς οργανισμοί, εταιρίες αλλά και απλοί χρήστες χρησιμοποιούν τέτοιου είδους υποδομές για αποθήκευση δεδομένων όπως επίσης και για χρησιμοποίηση υπολογιστικής ισχύος. Για τη διασφάλιση της αξιοπιστίας και τη βελτίωση των υπηρεσιών που προσφέρονται, η παρακολούθηση της υποδομής του νέφους είναι απαραίτητη. Λόγω του τεράστιου όγκου δεδομένων που παράγουν τέτοιου είδους υποδομές απαιτούνται σύγχρονες τεχνικές για την ανάλυση των δεδομένων αυτών και την εξαγωγή χρήσιμης γνώσης για το διαχειριστή.

Σκοπός της παρούσας διπλωματικής είναι η ανάπτυξη πλατφόρμας για την παρακολούθηση υποδομών υπολογιστικών νεφών από το διαχειριστή της υποδομής. Αρχικά, μελετώνται οι έννοιες των υπολογιστικών νεφών, Big Data και NoSQL βάσεων δεδομένων. Γίνεται ανάλυση των εργαλείων και των τεχνολογιών που χρησιμοποιήθηκαν καθώς και οι λόγοι επιλογής τους. Έπειτα αναλύεται η αρχιτεκτονική της πλατφόρμας. Τέλος υλοποιείται η πλατφόρμα στα πλαίσια της εργαστηριακής υποδομής και παρουσιάζονται τα αποτελέσματα της πειραματικής αυτής διαδικασίας. Στην εργασία παρουσιάζονται ο πλήρης κώδικας της πλατφόρμας καθώς και οι παραμετροποιήσεις των εργαλείων που χρησιμοποιήθηκαν.

Λέξεις κλειδιά

Big Data, NoSQL, Cloud, Monitoring, Data Analysis, Spark, Cassandra, OpenStack, Zabbix

Abstract

In the era of Big Data, Cloud Computing is particularly popular as organizations, companies and ordinary users use such infrastructures for data storage as well as computing power. To ensure the reliability and the improvement of the offered services, the monitoring of cloud infrastructure is essential. Due to the huge amount of data that produce this kind of infrastructures, modern techniques required to analyze the data and extract useful knowledge for the administrator.

The scope of this thesis is the development of a platform for monitoring cloud infrastructures by the infrastructure administrator. Initially, we study the concepts of Cloud Computing, Big Data and NoSQL databases. We analyze the tools and the technologies that were used and their reasons for choosing. Following is the analysis of the platform architecture. Finally, the platform is implemented within the laboratory infrastructure and results of this experimental procedure are presented. This thesis presents the complete code of the platform as well as the configuration of the tools used.

Keywords

Big Data, NoSQL, Cloud, Monitoring, Data Analysis, Spark, Cassandra, OpenStack, Zabbix

Ευχαριστίες

Με την παρούσα διπλωματική ολοκληρώνεται ένας κύκλος σπουδών στο τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π και μια από τις μεγαλύτερες εμπειρίες της ζωής μου και γι' αυτό το λόγο θα ήθελα να ευχαριστήσω όλους αυτούς που με βοήθησαν να φτάσω μέχρι εδώ.

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή Ιάκωβο Στ. Βενιέρη για την τιμή που μου έκανε να εκπονήσω την παρούσα διπλωματική υπό την επίβλεψη του. Η διπλωματική αυτή υπήρξε μια μοναδική ευκαιρία να ασχοληθώ με ένα τόσο σημαντικό και επίκαιρο θέμα και τον ευχαριστώ για αυτό.

Θα ήθελα επίσης να ευχαριστήσω τον υποψήφιο διδάκτορα Ανδρέα Καψάλη για την καθοδήγηση και την πολύτιμη βοήθεια του από την πρώτη έως και την τελευταία μέρα εκπόνησης της διπλωματικής μου εργασίας. Ο επαγγελματισμός και η ευγένεια του ήταν καθοριστικοί για την επίτευξη αυτού του εγχειρήματος.

Έπειτα θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδερφό μου οι οποίοι με στηρίζουν σε όλη μου τη ζωή, τους παππούδες μου για τις αξίες που μου μετέδωσαν, τον θείο μου Νίκο και τα ξαδέρφια μου για την αγάπη που μου μετέδωσαν για την τεχνολογία και τους υπολογιστές από πολύ μικρή ηλικία.

Τέλος θέλω να ευχαριστήσω την παγκόσμια ακαδημαϊκή κοινότητα και ιδιαίτερα του Ε.Μ.Π που με το επιστημονικό τους έργο και την ακαδημαϊκή τους στάση αλλάξανε τη ζωή μου και με έκαναν να γνωρίσω και να αγαπήσω την επιστήμη, τις αξίες και τα ιδανικά της.

Πίνακας περιεχομένων

Περίληψη 5
Abstract 7
Ευχαριστίες 9
Πίνακας Περιεχομένων 11
Κατάλογος Σχημάτων 13

1. Εισαγωγή.....15
1.1 Σκοπός της διπλωματικής.....15
1.2 Οργάνωση τόμου.....15

2. Το τοπίο των Big Data.....16

3. Υπόβαθρο.....19
3.1 Μεγάλα Δεδομένα.....19
3.2 NoSQL Βάσεις Δεδομένων…….....21
3.3 Υπολογιστικά Νέφη.....22

4. Εργαλεία και τεχνολογίες 24
4.1 OpenStack.....24
4.2 Zabbix.....25
4.3 Apache Cassandra.....25
4.4 Apache Spark.....28
4.5 Εργαλεία προγραμματισμού που χρησιμοποιήθηκαν.....30
Επισκόπηση κεφαλαίου.....32

5. Η αρχιτεκτονική της πλατφόρμας.....33
5.1 Η αρχιτεκτονική.....33
5.2 Ανάλυση των υπηρεσιών.....34

6. Πειραματική διαδικασία.....37
6.1 Υλοποίηση της πλατφόρμας.....37
6.1.1 Zabbix.....37
6.1.2 Zabbix - Cassandra Connector.....42
6.1.3 Log Cassandra Connector.....48
6.1.4 Spark & Spark Cassandra Analysis.....50
6.1.5 Data Show.....56
6.1.6 Cassandra.....58
6.2 Αποτελέσματα πειραματικής διαδικασίας.....59

Βιβλιογραφία 75
Βιβλιογραφία Σχημάτων 77
Παράρτημα Α.....78
Παράρτημα Β 80

Κατάλογος σχημάτων

Σχήμα 1. Το τοπίο των Big Data το 2016....16
Σχήμα 2. Πλατφόρμα της IBM για ανάλυση Big Data....17
Σχήμα 3. Πλατφόρμα της Hortonworks για ανάλυση Big Data....18
Σχήμα 4. Πλατφόρμα της Cloudera για ανάλυση Big Data....18
Σχήμα 5. Τομείς και χαρακτηριστικά των Big Data....19
Σχήμα 6. Μέτρηση λέξεων μέσω μοντέλου MapReduce....21
Σχήμα 7. Σύνοψη της αρχιτεκτονικής Cloud Computing....22
Σχήμα 8. Στρώματα Cloud Computing αναπαριστώμενα σε μια στοίβα....23
Σχήμα 9. OpenStack αρχιτεκτονική κύριων συστατικών μονάδων....24
Σχήμα 10. Apache Cassandra κλιμακωσιμότητα....25
Σχήμα 11. Μια γραμμή σε μια οικογένεια στηλών....27
Σχήμα 12. Μια γραμμή σε μια υπερ-οικογένεια στηλών....27
Σχήμα 13. Spark εναντίων Hadoop....28
Σχήμα 14. Δομικές μονάδες Spark....29
Σχήμα 15. Spark αρχιτεκτονική....29
Σχήμα 16. Cassandra και Spark μέσω Connector....32
Σχήμα 17. Σχηματικό διάγραμμα αρχιτεκτονικής της πλατφόρμας....34
Σχήμα 18. Υπηρεσία Παρακολούθησης....34
Σχήμα 19. Υπηρεσία Μεταφοράς Αρχείων....35
Σχήμα 20. Υπηρεσία Παρακολούθησης....35
Σχήμα 21. Υπηρεσία Διεπαφής με τον Διαχειριστή....36
Σχήμα 22. Zabbix εξυπηρετητές....38
Σχήμα 23 Zabbix ομάδα εξυπηρετητών....38
Σχήμα 24. Zabbix εφαρμογές....38
Σχήμα 25. Zabbix αντικείμενα εφαρμογής για τον επεξεργαστή....39
Σχήμα 26. Zabbix αντικείμενα εφαρμογής για τη μνήμη....39
Σχήμα 27. Zabbix αντικείμενα εφαρμογής για τον δίσκο....39
Σχήμα 28. Zabbix αντικείμενα εφαρμογής για το δίκτυο....40
Σχήμα 29. Zabbix Σκανδάλες α) 40
Σχήμα 30. Zabbix Σκανδάλες β) 41
Σχήμα 31. Zabbix script....41
Σχήμα 32. Σύμπλεγμα Zabbix....42
Σχήμα 33. Σύμπλεγμα Υπηρεσίας Παρακολούθησης....47
Σχήμα 34. Σύμπλεγμα Υπηρεσίας Μεταφοράς Αρχείων....48
Σχήμα 35. Σύμπλεγμα Υπηρεσίας Ανάλυσης Δεδομένων....50
Σχήμα 36. Προγραμματισμένη εκτέλεση των προγραμμάτων
της υπηρεσίας ανάλυσης δεδομένων.... 55
Σχήμα 37. Σύμπλεγμα Υπηρεσίας Διεπαφής με τον Διαχειριστή.... 55
Σχήμα 38. Spark κατά τη διάρκεια εκτέλεσης....58
Σχήμα 39. Ημερήσια ανάλυση της μονάδας Spark Cassandra Analysis....58
Σχήμα 40. Στοιχεία του Cassandra Cluster....58
Σχήμα 41. Μεγέθη βασικών πινάκων του Cassandra Cluster (σε bytes) 58
Σχήμα 42. Log Cassandra Connector σε εκτέλεση....59
Σχήμα 43. Zabbix Cassandra Connector σε εκτέλεση....59
Σχήμα 44. Εμφάνιση αποτελεσμάτων – results1_1 για 3 μέρες....59
Σχήμα 45. Εμφάνιση αποτελεσμάτων – results1_1 για 1 μέρα....60
Σχήμα 46. Εμφάνιση αποτελεσμάτων – results1_2....60
Σχήμα 47. Εμφάνιση αποτελεσμάτων – results1_3....60

Σχήμα 48. Εμφάνιση αποτελεσμάτων – results1_4.....61
Σχήμα 49. Εμφάνιση αποτελεσμάτων – results1_5.....61
Σχήμα 50. Εμφάνιση αποτελεσμάτων – results2.....62
Σχήμα 51. Εμφάνιση αποτελεσμάτων – results3.....62
Σχήμα 52. Εμφάνιση αποτελεσμάτων – results4.....62
Σχήμα 53. Εμφάνιση αποτελεσμάτων – results5.....63
Σχήμα 54. Εμφάνιση αποτελεσμάτων – results6.....63
Σχήμα 55. Εμφάνιση αποτελεσμάτων – results7.....63
Σχήμα 56. Εμφάνιση αποτελεσμάτων – results8.....63
Σχήμα 57. Μέση χρήση επεξεργαστή σε μία εβδομάδα.....64
Σχήμα 58. Μέση κίνηση εισερχόμενη στο δίκτυο σε μία εβδομάδα.....65
Σχήμα 59. Μέση κίνηση εξερχόμενη απ' το δίκτυο σε μία εβδομάδα.....66
Σχήμα 60. Μέση διαθέσιμη μνήμη σε μία εβδομάδα.....67
Σχήμα 61. Μέσος χρησιμοποιούμενος δίσκος σε μία εβδομάδα.....68
Σχήμα 62. Ενεργοποιημένες σκανδάλες και ποσοστό ενεργούς δράσης στη διάρκεια της ημέρας 69
Σχήμα 63. Μέση χρησιμοποίηση μνήμης για εφαρμογές με χρήση περισσότερη από 50%.....69
Σχήμα 64. Ποσοστό στη διάρκεια της μέρας για εφαρμογές με χρήση μνήμης περισσότερη από 30%.....70
Σχήμα 65. Μέση χρησιμοποίηση επεξεργαστή από εφαρμογές κατά τη διάρκεια μιας εβδομάδας.....71
Σχήμα 66. Εφαρμογές οι οποίες χρησιμοποίησαν περισσότερο από 30% τον επεξεργαστή και το ποσοστό στη διάρκεια της ημέρας που είχαν χρήση πάνω από 30%.....72
Σχήμα 67. Εφαρμογές οι οποίες χρησιμοποίησαν περισσότερο από 30% τον επεξεργαστή και το ποσοστό στη διάρκεια της ζωής τους που είχαν χρήση πάνω από 30%.....73
Σχήμα 68. Αριθμός προειδοποιήσεων από υπηρεσίες του OpenStack.....74
Σχήμα 69. Αριθμός σφαλμάτων από υπηρεσίες του OpenStack.....74

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της διπλωματικής

Ο όρος Μεγάλα Δεδομένα (Big Data [1]) περιγράφει δεδομένα που είναι μεγάλα σε όγκο, άναρχα δομημένα και παράγονται ανά σύντομα τακτά χρονικά διαστήματα. Η ανάλυση τέτοιου τύπου δεδομένων αποτελεί ένα σημαντικό πεδίο έρευνας στις μέρες μας.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη πλατφόρμας που θα διεξάγει ανάλυση δεδομένων σε τεράστια αρχεία και βάσεις δεδομένων που θα παράγονται από την λειτουργία και τη χρήση της υποδομής υπολογιστικού νέφους που βρίσκεται και αναπτύσσεται στο Εργαστήριο Ευφυών Επικοινωνιών και Δικτύων Ευρείας Ζώνης (ICBNet Lab [2]).

Σκοπός της ανάλυσης των δεδομένων αυτών είναι ο έλεγχος της λειτουργίας της υποδομής του εργαστηρίου μέσα από δημιουργία εξατομικευμένων προφίλ των χρηστών. Κατά αυτό το τρόπο θα εξασφαλίζεται η ομαλή λειτουργία της υποδομής και η συνεχής διαθεσιμότητα των υπηρεσιών που θα φιλοξενεί.

1.2 Οργάνωση τόμου

- Το κεφάλαιο 1 αποτελεί την εισαγωγή όπου σκιαγραφείται ο σκοπός της παρούσας εργασίας.

- Στο κεφάλαιο 2 παρουσιάζονται παρόμοιες πλατφόρμες ανάλυσης δεδομένων για υπολογιστικές υποδομές και σύγκριση με τη δική μας πλατφόρμα.

- Στο κεφάλαιο 3 γίνεται μια εισαγωγή σε έννοιες που διαδραμάτισαν σημαντικό ρόλο για την επιλογή των εργαλείων και των τεχνολογιών για την ανάπτυξη της πλατφόρμας.

- Στο κεφάλαιο 4 περιγράφονται οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν.

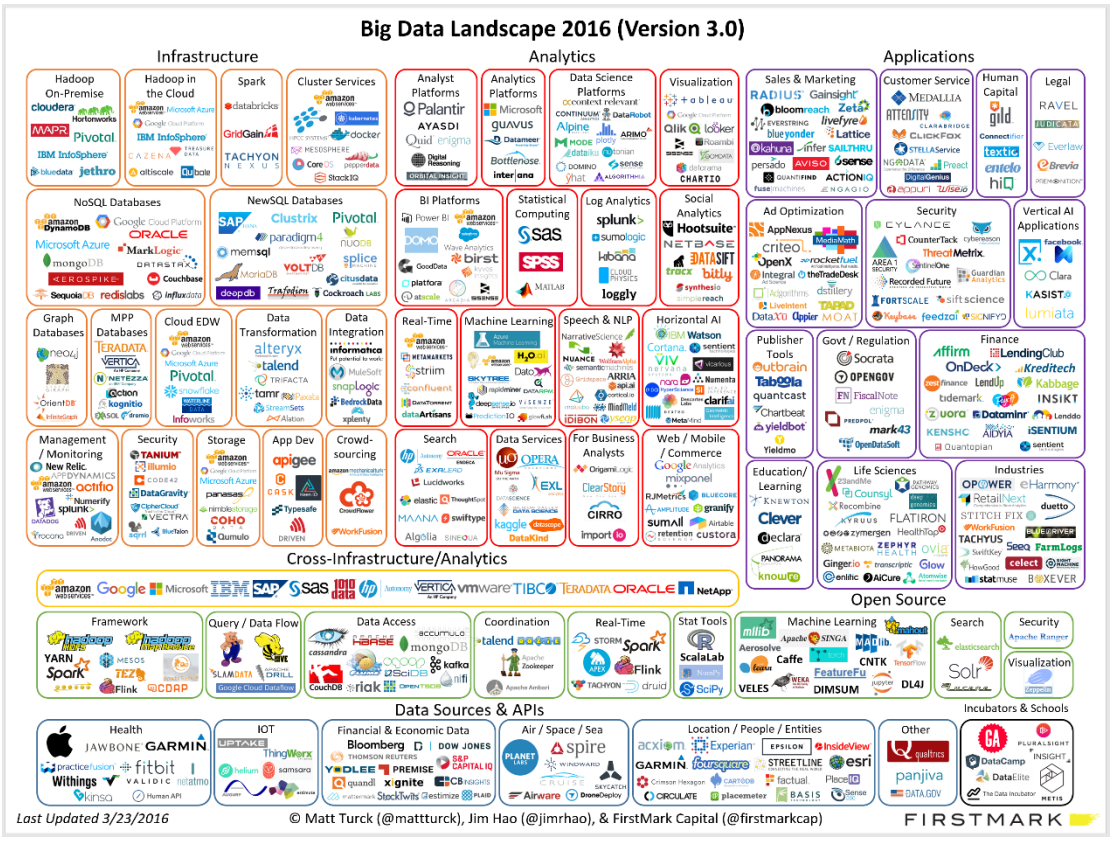
- Στο κεφάλαιο 5 αναλύεται η αρχιτεκτονική της πλατφόρμας.

- Το κεφάλαιο 6 αποτελεί υλοποίηση της πλατφόρμας, περιγραφή και αποτελέσματα της πειραματικής διαδικασίας.

Κεφάλαιο 2

Το τοπίο των Big Data

Στην εποχή μας ο τομέας των Big Data γνωρίζει τεράστια άνθηση. Ο όρος αυτός είναι γενικός και όπως φαίνεται από το παρακάτω σχήμα υπάρχουν διάφορες κατηγορίες και υποκατηγορίες. Η πλατφόρμα που σχεδιάστηκε η οποία θα παρουσιαστεί πλήρως στα επόμενα κεφάλαια αποτελεί μια υποδομή στην οποία γίνεται συλλογή, ανάλυση δεδομένων και παρουσίαση των αποτελεσμάτων που προκύπτουν.



Σχήμα 1. Το τοπίο του Big Data το 2016

Ανάλυση δημοφιλών λύσεων για big data analytics

Μέχρι πρόσφατα η ανάλυση Big Data γινόταν κατά κύριο λόγο μέσω του Apache Hadoop[3].Συγκριτικά με το 2015, το 2016 είναι μια χρονιά όπου οι περισσότερες εταιρίες πρόσθεσαν το Apache Spark [4] για κατανεμημένη επεξεργασία στη μνήμη έναντι στον σκληρό δίσκο με αποτέλεσμα την ραγδαία αύξηση στην ταχύτητα επεξεργασίας.

Οι λύσεις που θα αναλυθούν θα είναι γενικού σκοπού ανάλυσης δεδομένων αλλά και πλατφόρμες οι οποίες είναι παρόμοιες με αυτή που δημιουργήθηκε.

Splunk [5]

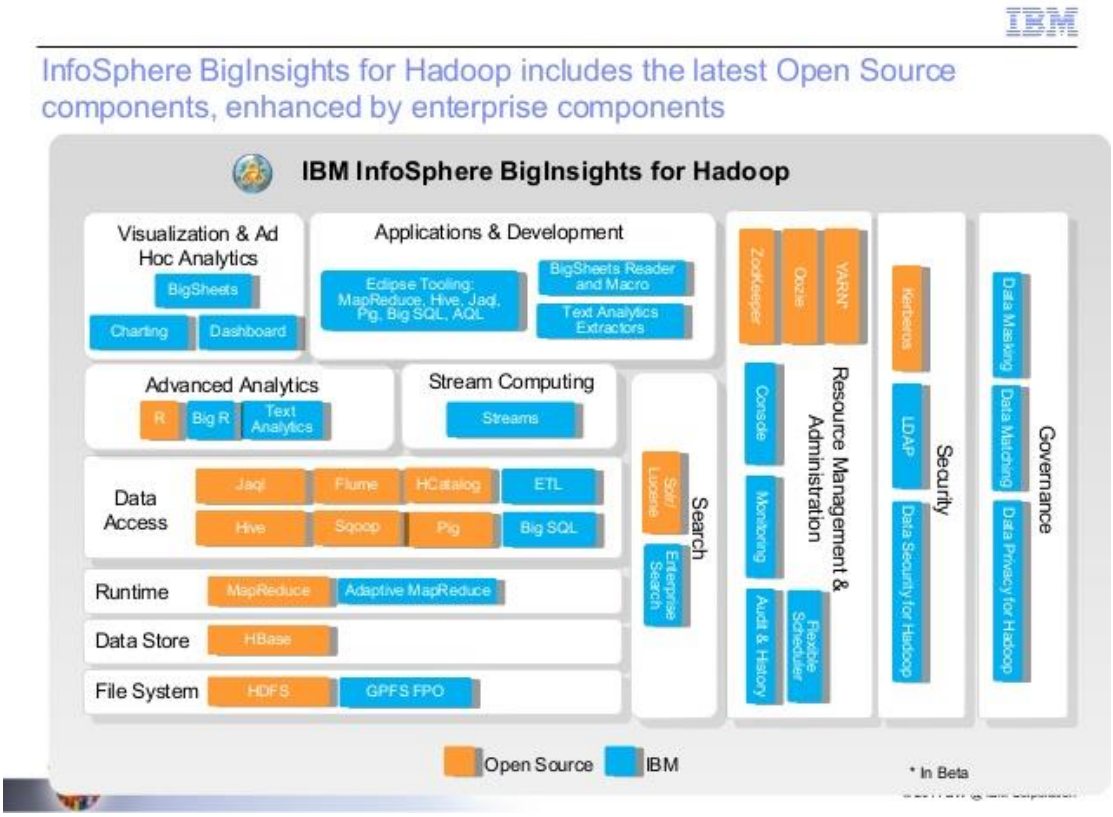
Το Splunk απευθύνεται σε επιχειρήσεις που παράγουν πολλά δικά τους δεδομένα μέσα από τις δικές του μηχανές. Είναι ένα λογισμικό για αναζήτηση, παρακολούθηση και ανάλυση δεδομένων. Το 2013 προστέθηκε υποστήριξη του Hadoop για υποστήριξη μεγάλου όγκου δεδομένων.

ELK stack [6]

Αποτελείται από τρεις τεχνολογίες:
Elasticsearch για αναζήτηση και ανάλυση δεδομένων
Logstash για logs
Kibana για εικονοποίηση των αποτελεσμάτων

IBM Big Data Analytics [7]

Όπως και άλλες big data εταιρίες η IBM χτίζει την πλατφόρμα της στο Apache Hadoop οπότε είναι γρήγορο, οικονομικό και ανοιχτού κώδικα. Μέσω του προϊόντος BigInsights δίνει στις επιχειρήσεις τη δυνατότητα επεξεργασίας δομημένων και μη δεδομένων. Μέσω του InfoSphere Streams επιτρέπεται η «αιχμαλώτιση» και η επεξεργασία ζωντανών δεδομένων. Επίσης προσφέρεται εικονοποίηση των αποτελεσμάτων.



Σχήμα 2. Πλατφόρμα της IBM για ανάλυση Big Data

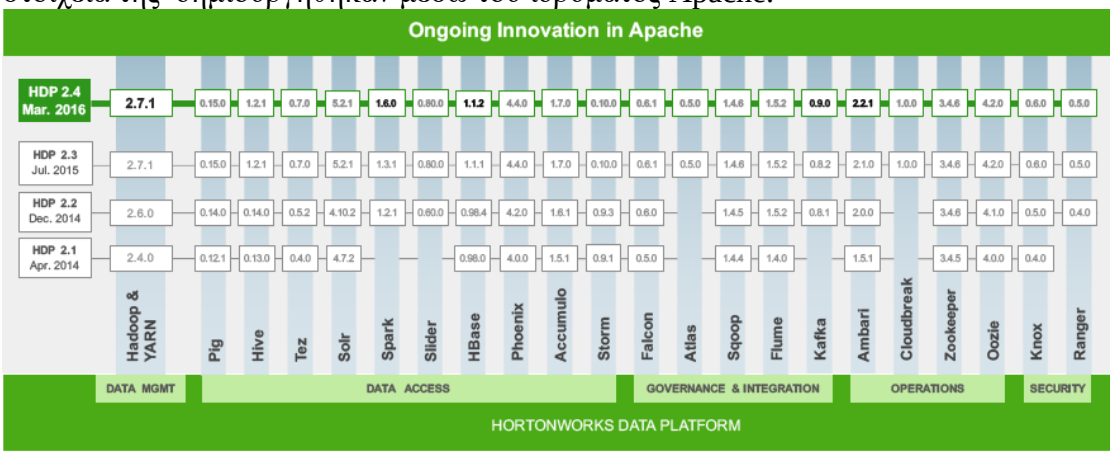
Microsoft

Η λύση της Microsoft ονομάζεται HDInsight [8] στηρίζεται στο Hortonworks Data Platform αλλά είναι προσαρμοσμένη να λειτουργεί με το Azure, που αποτελεί το δικό

της Cloud και τον SQL Server. Μεγάλο πλεονέκτημα αποτελεί η ενσωμάτωση στο Excel.

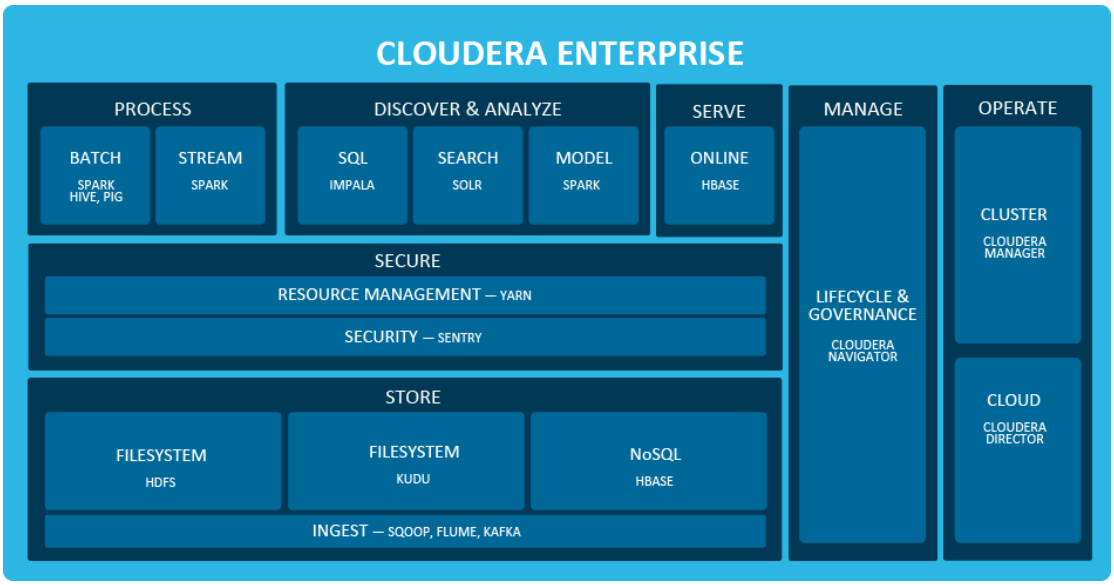
Hortonworks Data Platform [9]

Η HDP σε αντίθεση με άλλες τις πλατφόρμες στηρίζεται σε ανοιχτό κώδικα και όλα τα στοιχεία της δημιουργήθηκαν μέσω του ιδρύματος Apache.



Σχήμα 3. Πλατφόρμα της Hortonworks για ανάλυση Big Data

Cloudera CDH [10]



Σχήμα 4. Πλατφόρμα της Cloudera για ανάλυση Big Data

Σύγκριση χαρακτηριστικών με τη δική μας πλατφόρμα

Το Splunk και το ELK μοιάζουν περισσότερο με την πλατφόρμα που δημιουργήθηκε. Οι άλλες λύσεις αποτελούν λύσεις για ανάλυση δεδομένων γενικού σκοπού. Η δική μας πλατφόρμα χρησιμοποιεί το Apache Spark σε συνδυασμό με την Apache Cassandra για την ανάλυση Big Data. Σε αντίθεση με τις άλλες πλατφόρμες η Cassandra ως βάση δεδομένων αποτελεί τη λύση με την καλύτερη κλιμάκωση. Σε σύγκριση με το Splunk και το ELK η πλατφόρμα μας χρησιμοποιώντας το Spark αποτελεί μια πολύ γρηγορότερη λύση για περιπτώσεις τεράστιου όγκου δεδομένων.

Κεφάλαιο 3

Υπόβαθρο

Είναι αναγκαίο να αναφερθούν και να αναλυθούν όροι οι οποίοι είναι καθοριστικοί για την επιλογή των εργαλείων που θα χρησιμοποιηθούν για τον αρχιτεκτονικό σχεδιασμό της πλατφόρμας προτού γίνει ανάλυση της.

3.1 Μεγάλα δεδομένα

Σχήμα 5. Τομείς και χαρακτηριστικά των Big Data

Μεγάλα Δεδομένα (Big Data) είναι ένας όρος για συλλογές δεδομένων οι οποίες είναι πολύ μεγάλες ή σύνθετες και οι παραδοσιακοί τρόποι επεξεργασίας είναι ανεπαρκείς. Οι προκλήσεις περιλαμβάνουν ανάλυση, συλλογή, επιμέλεια δεδομένων, αναζήτηση, διαμοιρασμό, αποθήκευση, μεταφορά, εικονοποίηση, ερωτήματα(quering), ανανέωση(update) και ιδιωτικότητα πληροφοριών. Ο όρος Big Data συχνά αναφέρεται απλά στη χρήση προγνωστικών αναλύσεων ή στη χρησιμοποίηση άλλων προχωρημένων τεχνικών για την εξαγωγή αξίας από τα δεδομένα και σπάνια στον όγκο των δεδομένων. Η ακρίβεια στα Big Data μπορεί να οδηγήσει σε καλύτερη λήψη αποφάσεων και καλύτερες αποφάσεις οδηγούν σε μεγαλύτερη λειτουργική αποτελεσματικότητα, μείωση κόστους και μείωση ρίσκου.

Χαρακτηριστικά των Μεγάλων Δεδομένων

Τα Big Data μπορούν να περιγραφούν από τα εξής ακόλουθα χαρακτηριστικά τα οποία έχουν επεκταθεί σε 5V από 3V ,που είχε αναφερθεί ήδη στο μοντέλο αυτό από το 2001 ο αναλυτής Doug Laney [11].

- **Όγκος (Volume)**
  Η ποσότητα παραγωγής και αποθήκευσης δεδομένων. Το μέγεθος των δεδομένων καθορίζει την αξία, την ενδεχόμενη διορατικότητα όπως επίσης και αν μπορεί πράγματι να θεωρηθεί Big Data ή όχι.

- **Ποικιλία (Variety)**
  Ο τύπος και η φύση των δεδομένων. Αυτό βοηθάει τους ανθρώπους που τα αναλύουν να χρησιμοποιήσουν αποτελεσματικά την προκύπτουσα διορατικότητα.

- **Ταχύτητα (Velocity)**
  Η ταχύτητα με την οποία παράγεται και επεξεργάζεται ο όγκος των δεδομένων ώστε να ικανοποιήσουν τις ανάγκες και τις προκλήσεις που βρίσκονται στο μονοπάτι της ανάπτυξης.

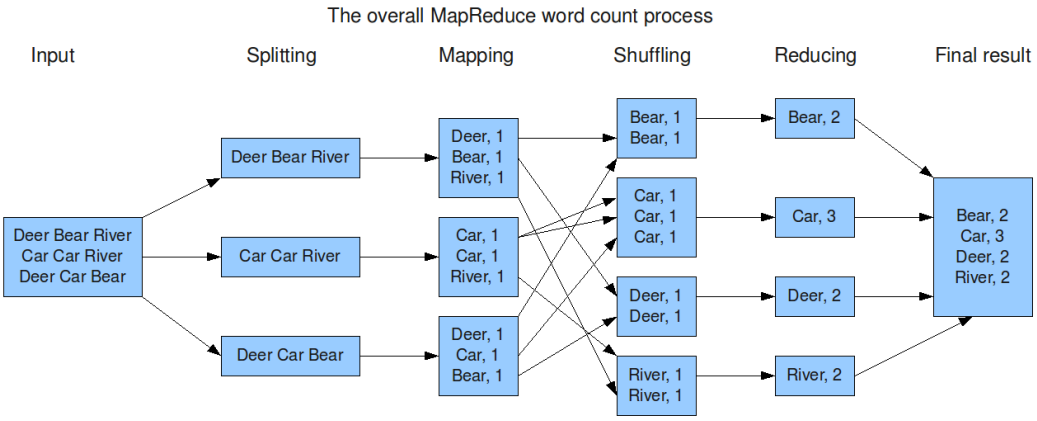
- **Μεταβλητότητα (Variability)**
  Η ασυνέπεια της συλλογής δεδομένων μπορεί να παρεμποδίσει τις διαδικασίες για το χειρισμό και τη διαχείριση του.

- **Φιλαλήθεια (Veracity)**
  Η ποιότητα των δεδομένων που έχουν καταγραφεί μπορεί να ποικίλει σε μεγάλο βαθμό επηρεάζοντας την σε ακρίβεια ανάλυση.

MapReduce

Το MapReduce[12] είναι ένα προγραμματιστικό μοντέλο και μια σχετική υλοποίηση για την επεξεργασία και τη δημιουργία μεγάλων συνόλων δεδομένων. Οι χρήστες καθορίζουν μια συνάρτηση map η οποία επεξεργάζεται ένα ζεύγος κλειδιού-τιμής (key-value pair) για να δημιουργηθεί ένα σύνολο ενδιάμεσων key-value ζευγών και μια συνάρτηση reduce που συγχωνεύει όλες τις ενδιάμεσες τιμές η οποίες μοιράζονται το ίδιο κλειδί.

Τα προγράμματα που είναι γραμμένα σε αυτό το συναρτησιακού προγραμματισμού στυλ παραλληλοποιούνται αυτόματα και εκτελούνται σε ένα μεγάλο σύμπλεγμα υπολογιστών. Το σύστημα χρόνου εκτέλεσης(run-time system) αναλαμβάνει την διχοτόμηση των δεδομένων εισόδου, τον προγραμματισμό εκτέλεσης του προγράμματος σε μια σειρά από μηχανές, το χειρισμό μηχανικών σφαλμάτων και την διαχείριση της επικοινωνίας μεταξύ των μηχανημάτων.



Σχήμα 6. Μέτρηση λέξεων μέσω μοντέλου MapReduce

3.2 NoSQL Βάσεις Δεδομένων

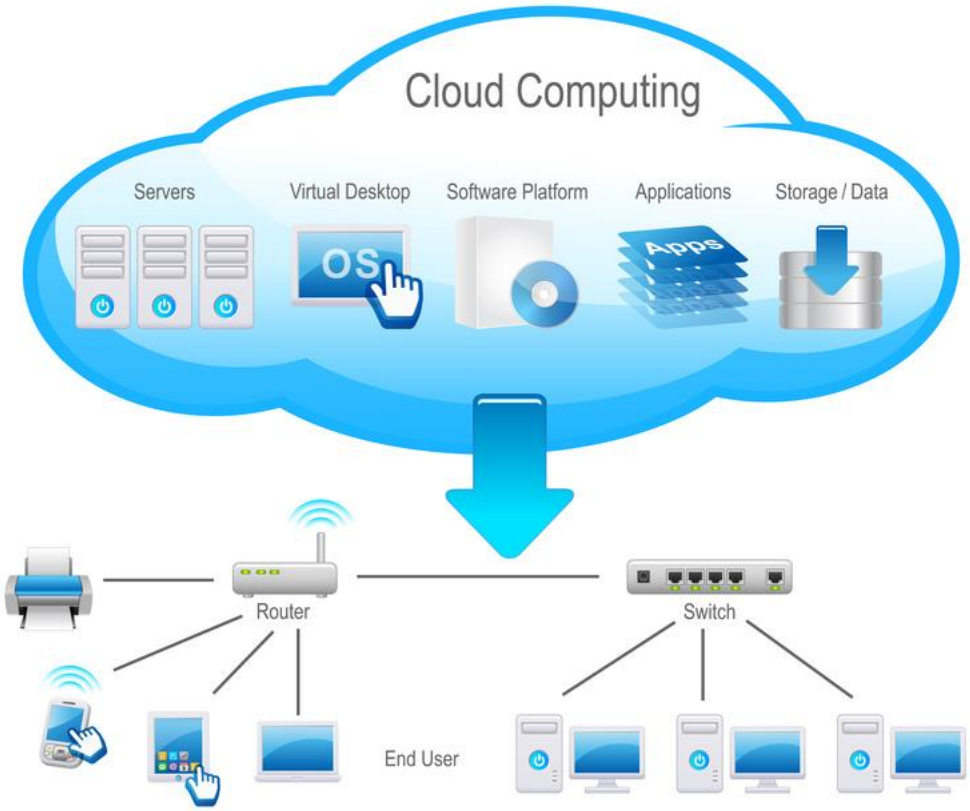
Μια NoSQL[13] βάση δεδομένων παρέχει ένα μηχανισμό αποθήκευσης και ανάκτησης δεδομένων έχοντας ως μοντέλο διαφορετικό των πινακοειδών σχέσεων που συναντώνται σε σχεσιακές βάσεις δεδομένων. Η τεχνολογία αυτή είναι διαθέσιμη ήδη από τα τέλη του 1960 χωρίς τη συγκεκριμένη ονομασία αλλά οι ανάγκες του Web 2.0 στον 21ο αιώνα καθιέρωσαν τον όρο.

Τα κίνητρα για αυτή την προσέγγιση περιλαμβάνουν: απλότητα στο σχεδιασμό, οριζόντια κλιμάκωση σε συστάδες υπολογιστών και καλύτερο έλεγχο διαθεσιμότητας. Οι δομές δεδομένων που χρησιμοποιούνται στις NoSQL είναι διαφορετικές απ' τις σχεσιακές βάσεις με αποτέλεσμα κάποιες λειτουργίες να γίνονται πολύ γρηγορότερα.

Οι NoSQL βάσεις δεδομένων χωρίζονται στις εξής κατηγορίες:

- Στήλες (Column)
- Έγγραφα (Document)
- Κλειδί-Τιμή (key-value)
- Γράφοι (Graph)
- Πολυ-μοντέλο (Multi-model)

3.3 Υπολογιστικά νέφη



Σχήμα 7. Σύνοψη της αρχιτεκτονικής Cloud Computing

.

Τα υπολογιστικά νέφη(Cloud Computing[14]) αναδύθηκαν ως ένα νέο πρότυπο για την παροχή διαφοροποιημένων πόρων, όπως η χρήση υπολογιστικής δύναμης, η αποθήκευση δεδομένων και η χρησιμοποίηση εύρους ζώνης δικτύου.

Αυτοί οι πόροι προσφέρονται στους χρήστες ως υπηρεσίες μέσω εικονικότητας(virtualization). Οι χρήστες μισθώνουν τις υπηρεσίες κατά παραγγελία οι οποίες κλιμακώνονται ελαστικά σε σύντομο χρονικό διάστημα ανάλογα με τις ανάγκες τους, πληρώνοντας ανάλογα με τις εκάστοτε ανάγκες. Αυτό το παράδειγμα έφερε επανάσταση στον τρόπο προσέγγισης της διαχείρισης πόρων με αποτέλεσμα την μεταφορά ιδιοκτησίας και διαχείρισης πόρων από το «εντός οίκου» μοντέλο, στον πάροχο νέφους.

Αυτή η τεχνολογική επανάσταση έχει ιδιαίτερο αντίκτυπο στην εποχή των Big Data όπου εμφανίζεται ένας αυξημένος αριθμός εφαρμογών που έχουν ανάγκη κλιμάκωσης για να καλύψουν τις αποθηκευτικές και επεξεργαστικές προκλήσεις.

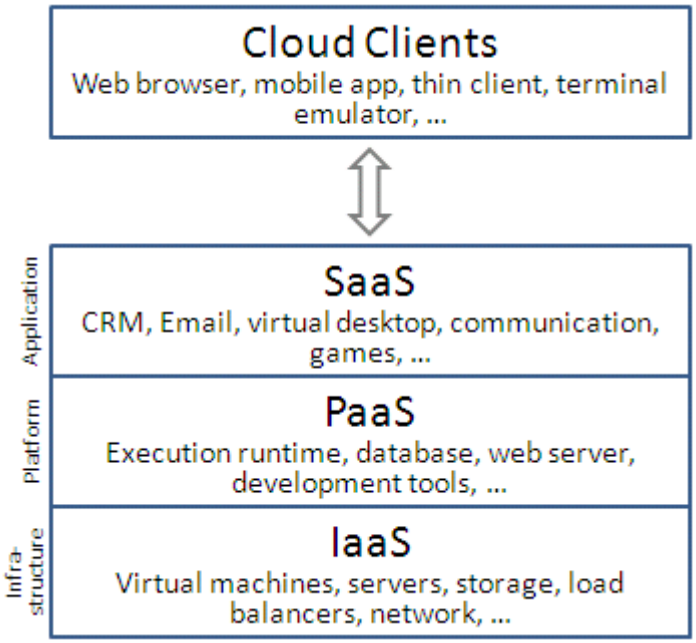
Επισκόπηση

Το υπολογιστικό νέφος επιτρέπει στους χρήστες να επικεντρωθούν στην εξόρυξη αξίας, στην ενοικίαση και κλιμάκωση των υπηρεσιών για βέλτιστη αξιοποίηση πόρων. Από την πλευρά τους οι πάροχοι διαχειρίζονται τους πόρους και προσφέρουν διαφορετικά επίπεδα ελέγχου στους χρήστες.

Όσον αφορά την προσβασιμότητα τους τα νέφη χωρίζονται σε ιδιωτικά και δημόσια ή και σε υβριδικά-μια μίξη δηλαδή ιδιωτικών και δημοσίων.

Μια άλλη μορφή ταξινόμησης γίνεται με βάση την λειτουργικότητα της κάθε υπηρεσίας που προσφέρεται. Οι προκύπτουσες κατηγορίες εμφανίζονται με την ένδειξη «ως Υπηρεσία»(as a Service) και παρέχουν τις παρακάτω λειτουργίες:

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



Σχήμα 8. Στρώματα Cloud Computing αναπαριστώμενα σε μια στοίβα

Κεφάλαιο 4

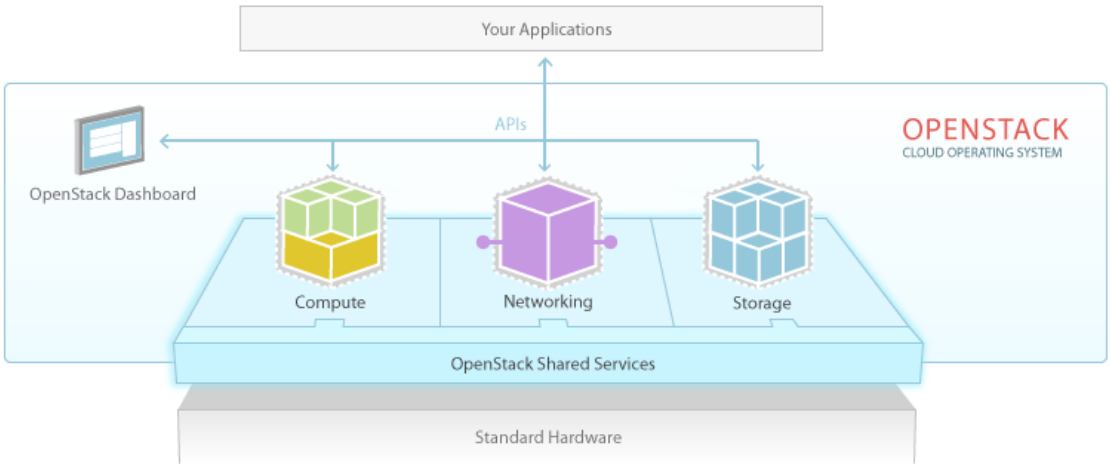
Εργαλεία και τεχνολογίες

4.1 OpenStack

Το OpenStack[15] είναι μια δωρεάν και ανοιχτού κώδικα Cloud Computing πλατφόρμα λογισμικού, κυρίως αναπτύχθηκε ως IaaS. Η πλατφόρμα αποτελείται από αλληλένδετα μέρη που ελέγχουν hardware pools επεξεργασίας, αποθήκευσης και δικτυακών πηγών σε Data Centers. Οι χρήστες διαχειρίζονται την πλατφόρμα είτε μέσω ενός δικτυακά-βασισμένου (web-based) ταμπλό, είτε μέσω εργαλείων της γραμμής εντολών ή μέσω ενός RESTful API [16].

Υπηρεσίες του πυρήνα του OpenStack

- Compute (Nova)
- Networking (Neutron)
- Block Storage (Cinder)
- Identity (Keystone)
- Image (Glance)
- Object Storage (Swift)



Σχήμα 9. OpenStack αρχιτεκτονική κύριων συστατικών μονάδων

4.2 Zabbix

Το Zabbix[17] είναι ένα ανοιχτού κώδικα πρόγραμμα παρακολούθησης (monitoring) για δίκτυα και εφαρμογές, δημιουργήθηκε από τον Alexei Vladishev. Έχει σχεδιαστεί να παρακολουθεί και να ανιχνεύει την κατάσταση των διάφορων υπηρεσιών δικτύου, servers και άλλο υλικό (hardware) του δικτύου. Η τελευταία έκδοση είναι η 3.0.3, στις 18 Μαΐου του 2016.

Περιλαμβάνει πολλές ξεχωριστές ενότητες:
- Servers
- Agents
- Frontend
- Proxy

Επίσης διαθέτει το Zabbix API[18] που χρησιμοποιεί το JSON-RPC 2.0[19] πρωτόκολλο, μέσω του οποίου μπορούν να αντληθούν στοιχεία του Zabbix δίνοντας τη δυνατότητα αξιοποίησης τους.

4.3 Apache Cassandra

Η Apache Cassandra[20] είναι μια δωρεάν και ανοιχτού κώδικα κατανεμημένη βάση δεδομένων σχεδιασμένη ώστε να διαχειρίζεται μεγάλες ποσότητες δεδομένων σε πολλούς εξυπηρετητές παρέχοντας υψηλή διαθεσιμότητα χωρίς κανένα σημείο αποτυχίας. Προσφέρει ισχυρή υποστήριξη για συμπλέγματα σε πολλαπλά κέντρα δεδομένων με ασύγχρονη αντιγραφή επιτρέποντας υψηλή απόδοση και χαμηλή καθυστέρηση για τους χρήστες.

Αρχικά αναπτύχθηκε στο Facebook για να κάνει δυνατή την αναζήτηση στην υπηρεσία Messanger της εταιρίας, αρχική εμφάνιση ως πρόγραμμα ανοιχτού κώδικα έκανε το 2008, το 2009 ενσωματώθηκε στο ίδρυμα Apache ενώ το 2010 έγινε ένα κορυφαίου επιπέδου πρόγραμμα. Τελευταία σταθερή έκδοση της Cassandra είναι η 3.4 με έκδοση της στις 8 Μαρτίου του 2016.

Η Cassandra αποτελεί μια NoSQL βάση δεδομένων. Το 2012 το πανεπιστήμιο του Τορόντο στον Καναδά ερευνητές μελέτησαν NoSQL συστήματα και κατέληξαν ότι σε όρους κλιμακωσιμότητας ο ξεκάθαρος νικητής είναι η Cassandra καθώς επιτυγχάνει την υψηλότερη απόδοση για το μέγιστο αριθμό των κόμβων σε όλα τα πειράματα-αν και αυτό έρχεται με το υψηλό τίμημα των μεγάλων καθυστερήσεων(high latency) σε εγγραφή και ανάγνωση.

Σχήμα 10. Apache Cassandra κλιμακωσιμότητα

Κύρια χαρακτηριστικά

- **Αποκεντρωμένη (Decentralized)**

Κάθε κόμβος του συμπλέγματος έχει τον ίδιο ρόλο. Δεν υπάρχει σημείο αποτυχίας (no single point of failure). Τα δεδομένα είναι κατανεμημένα σε όλη τη συστάδα (οπότε κάθε κόμβος περιέχει διαφορετικά δεδομένα) αλλά δεν υπάρχει αφέντης (master) και κάθε κόμβος μπορεί να εξυπηρετήσει οποιοδήποτε αίτημα.

- **Υποστηρίζει αντιγραφή και μεταξύ πολλαπλών κέντρων αντιγραφή (Supports replication and multi data center replication)**

Οι στρατηγικές αντιγραφής είναι διαμορφώσιμες. Η Cassandra είναι σχεδιασμένη ως κατανεμημένο σύστημα για την ανάπτυξη ενός μεγάλου αριθμού κόμβων και πολλαπλών κέντρων δεδομένων(data centers). Τα βασικά χαρακτηριστικά της κατανεμημένης αρχιτεκτονικής είναι προσαρμοσμένα ειδικά για την ανάπτυξη πολλαπλών κέντρων δεδομένων, για εφεδρεία, για προστασία απ' τις αποτυχίες και για αποκατάσταση καταστροφών.

- **Κλιμακωσιμότητα (Scalability)**

Η ταχύτητα και το εύρος ανάγνωσης και εγγραφής κλιμακώνονται γραμμικά καθώς νέα μηχανήματα προστίθενται, χωρίς χρόνο αργίας (downtime) και διακοπή εφαρμογών.

- **Ανεκτικότητα σφαλμάτων (Fault-tolerant)**

Τα δεδομένα αντιγράφονται αυτόματα σε πολλαπλούς κόμβους για ανεκτικότητα σφαλμάτων. Αντιγραφή μεταξύ πολλαπλών κέντρων επιτρέπεται. Οι αποτυχημένοι (failed) κόμβοι μπορούν να αντικατασταθούν χωρίς χρόνο αργίας.

- **Ρυθμιζόμενη συνοχή (Tunable consistency)**

Εγγραφές και αναγνώσεις προσφέρουν ένα ρυθμιζόμενο επίπεδο συνοχής σε όλη τη διαδρομή από το «οι εγγραφές δεν αποτυγχάνουν ποτέ» έως «τα μπλοκ των αντιγράφων να είναι αναγνώσιμα» με το επίπεδο απαρτίας στο κέντρο.

- **Υποστήριξη MapReduce**

Η Cassandra έχει Hadoop ενσωμάτωση με MapReduce υποστήριξη. Υπάρχει υποστήριξη για Apache Pig[21], Apache Hive[22] και Apache Spark.

- **Γλώσσα ερωτημάτων (Query language)**

Η Cassandra εισάγει την "CQL" (Cassandra Query Language) μια γλώσσα που μοιάζει με την «Σχεσιακή Γλώσσα Ερωτημάτων» (SQL). Αποτελεί μια προγραμματιστική διεπαφή για την πρόσβαση στην Cassandra προσφέροντας ένα επίπεδο αφαίρεσης, το οποίο κρύβει λεπτομέρειες της δομής και προσφέρει εγγενή για συλλογές και άλλες κοινές κωδικοποιήσεις. Οι οδηγοί (drivers) για γλώσσες είναι διαθέσιμοι για Java[23], Python[24] κτλ.

Το μοντέλο δεδομένων

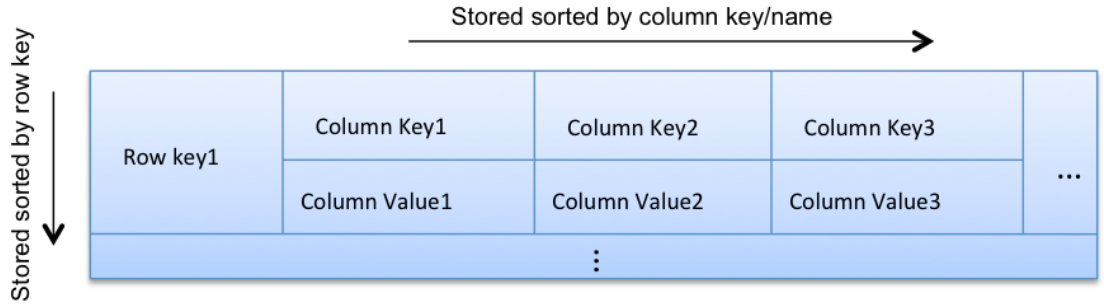
Το μοντέλο που χρησιμοποιείται είναι ένα υβριδικό μεταξύ ενός key-value και column-oriented μοντέλου συστήματος διαχείρισης. Το μοντέλο των δεδομένων είναι μία κατανεμημένη κατά σειρά αποθήκη με ρυθμιζόμενη συνοχή. Οι γραμμές οργανώνονται σε πίνακες. Το πρώτο συστατικό ενός πίνακα είναι το κλειδί διαμοιρασμού. Μέσα στη δομή αυτή οι γραμμές συγκεντρώνονται απ' τις υπολειπόμενες στήλες του κλειδιού. Άλλες στήλες μπορούν να βρεθούν ξεχωριστά απ' το πρωτεύον κλειδί.

Οι πίνακες μπορούν να δημιουργηθούν, να καταργηθούν ή να μεταβληθούν κατά το χρόνο εκτέλεσης χωρίς να εμποδίζονται οι ενημερώσεις και τα ερωτήματα.

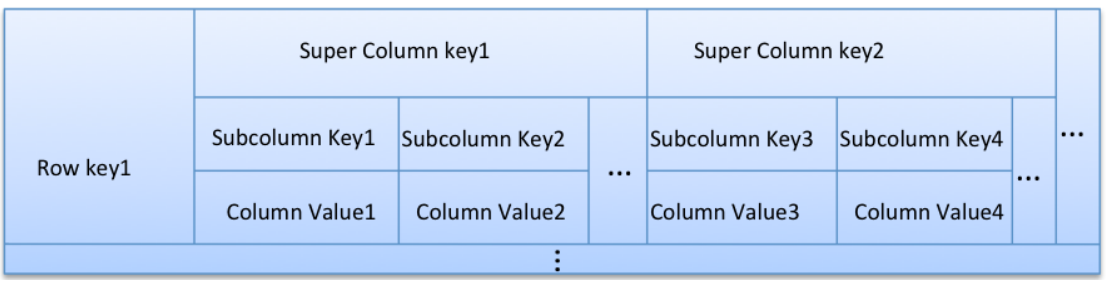
Δεν μπορούν να γίνουν ενώσεις (joins) ή υποερωτήματα (subqueries), αντίθετα δίνεται έμφαση στην απομαλοποίηση (denormalization) μέσω χαρακτηριστικών όπως οι συλλογές.

Μια οικογένεια στηλών (column family) μοιάζει με έναν πίνακα σε μια σχεσιακή βάση δεδομένων (RDBMS). Κάθε σειρά μοναδικά αναγνωρίζεται από ένα κλειδί σειράς. Κάθε γραμμή έχει πολλαπλές στήλες οι οποίες έχουν όνομα, τιμή και χρονοσφραγίδα (timestamp). Σε αντίθεση με μια σχεσιακή βάση δεδομένων διαφορετικές γραμμές στην ίδια οικογένεια στηλών δεν χρειάζεται να μοιράζονται τον ίδιο αριθμό στηλών και μια στήλη μπορεί να προστεθεί σε μια ή πολλαπλές σειρές ανά πάσα στιγμή.

Κάθε κλειδί στην Cassandra αντιστοιχεί σε μια τιμή το οποίο είναι ένα αντικείμενο, έχει τιμές ως στήλες και οι στήλες ομαδοποιούνται σε ομάδες οι οποίες αποκαλούνται οικογένειες στηλών. Έτσι κάθε κλειδί αναγνωρίζει μια γραμμή μεταβλητού αριθμού στοιχείων. Αυτές οι οικογένειες στηλών μπορούν να θεωρηθούν πίνακες. Ένας πίνακας στην Cassandra είναι ένας κατανεμημένος πολυδιάστατος πίνακας με δείκτη ένα κλειδί. Επιπλέον οι εφαρμογές μπορούν να καθορίσουν τη σειρά ταξινόμησης των στηλών μέσα σε μια απλή οικογένεια στηλών ή σε μια υπερ-οικογένεια στηλών(Super Column Family).



Σχήμα 11. Μια γραμμή σε μια οικογένεια στηλών



Σχήμα 12. Μια γραμμή σε μια υπερ-οικογένεια στηλών

4.4 Apache Spark

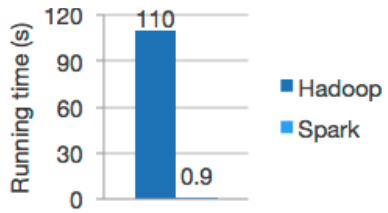
Το Apache Spark είναι ένα προγραμματιστικό πλαίσιο ανοιχτού κώδικα που υποστηρίζει την κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων. Αρχικά αναπτύχθηκε στο πανεπιστήμιο Berkley της California, στο εργαστήριο AMPLAB [25]-ενώ αργότερα ο πυρήνας του κώδικα δωρίστηκε στο ίδρυμα Apache Software Foundation όπου και συντηρείται έκτοτε. Η αρχική διάθεση έγινε στις 30 Μαΐου του 2014 και έκτοτε είναι εκ των κορυφαίων στη λίστα του ιδρύματος. Η τελευταία σταθερή έκδοση είναι η 1.6.1 όπου δημοσιεύθηκε στις 9 Μαρτίου του 2016.

Επισκόπηση

Το Spark παρέχει μια διεπαφή για τον προγραμματισμό υπολογιστικών συστάδων με «σιωπηρό» παραλληλισμό δεδομένων και ανεκτικότητα στα σφάλματα υλικού-ακόμα και αν κάποιος υπολογιστής παρουσιάσει πρόβλημα η εκτέλεση του προγράμματος δεν θα σταματήσει αλλά θα διαμοιράσει την εκτέλεση στους διαθέσιμους κόμβους.

Έχει ως κεντρικό άξονα μια δομή δεδομένων η οποία καλείται resilient distributed dataset (RDD [26]) ,ένα πολυσύνολο-μόνο για ανάγνωση-δεδομένων διαμοιρασμένων στο σύμπλεγμα των υπολογιστών, το οποίο διατηρείται σε ένα ανεκτικό στα σφάλματα τρόπο. Αναπτύχθηκε ως απάντηση στους περιορισμούς του μοντέλου MapReduce το οποίο ακολουθεί μια συγκεκριμένη ροή δεδομένων ως δομή στα κατανεμημένα προγράμματα: Τα MapReduce προγράμματα διαβάζουν τα δεδομένα εισόδου απ' το δίσκο εφαρμόζουν μια συνάρτηση σε όλα τα δεδομένα, ομαδοποιούν-«μειώνουν» τα αποτελέσματα και τα αποθηκεύουν στο δίσκο.

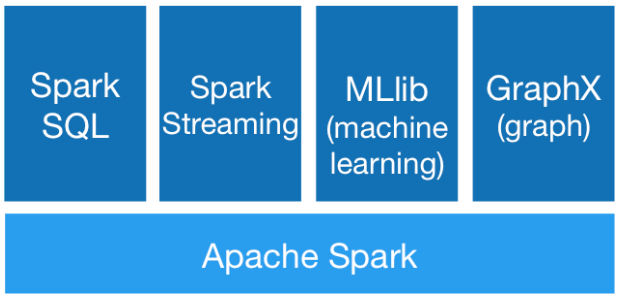
Η διαθεσιμότητα των RDD διευκολύνει την εφαρμογή τόσο των επαναληπτικών αλγορίθμων, όπου επισκέπτονται το σύνολο των δεδομένων πολλές φορές σε ένα βρόχο, όσο και την διαδραστική-διερευνητική ανάλυση δεδομένων, την για παράδειγμα επαναλαμβανόμενη εκτέλεση ερωτημάτων σε μια βάση δεδομένων. Η απόδοση τέτοιων εφαρμογών (σε σύγκριση με το Apache Hadoop ,μια διάσημη MapReduce υλοποίηση) μπορεί να αυξηθεί κατά τάξεις μεγέθους.



Σχήμα 13. Spark εναντίον Hadoop

Το Spark απαιτεί ένα διαχειριστή συμπλέγματος και ένα κατανεμημένο σύστημα αποθήκευσης. Ως διαχειριστή συμπλέγματος το Spark υποστηρίζει αυτόνομο(standalone) διαχειριστή, Hadoop Yarn [27] και Apache Mesos [28]. Ως κατανεμημένο σύστημα αποθήκευσης υποστηρίζεται μια ευρεία γκάμα συστημάτων, συμπεριλαμβανομένων των Hadoop Distributed File System (HDFS [29]), MapR File System (MapR-FS [30]), Cassandra, OpenStack Swift [31], Amazon S3 [32], Kudu [33] ή άλλων προσαρμοσμένων(custom) λύσεων.

Αποτελείται από τις εξής δομικές μονάδες:

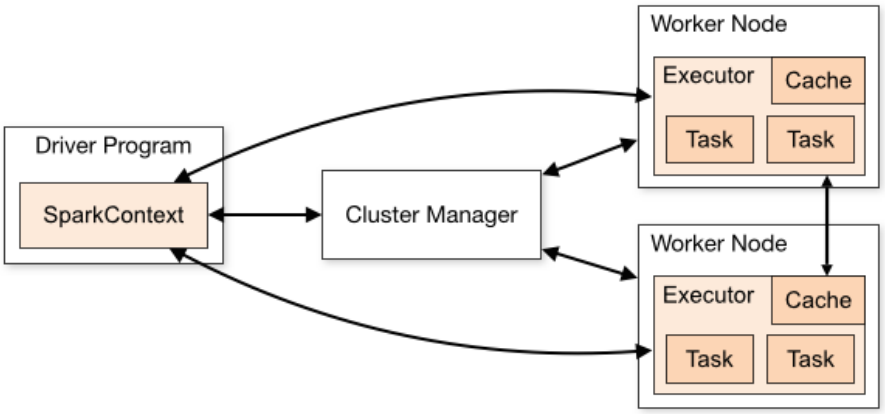


Σχήμα 14. Δομικές μονάδες Spark

Apache Spark / Core

Το Spark Core είναι ο θεμέλιος λίθος του συνολικού πρότζεκτ. Παρέχει κατανεμημένη αποστολή εργασιών, χρονοδρομολόγηση και βασικές λειτουργίες εισόδου εξόδου οι οποίες γίνονται διαθέσιμες μέσω των προγραμματιστικών διεπαφών (σε Java, Python, Scala [34] και R [35]) με κέντρο το RDD. Αυτή η διεπαφή καθρεφτίζει ένα λειτουργικό-ανώτερης τάξης μοντέλο προγραμματισμού:
Ένα πρόγραμμα που ονομάζεται «Οδηγός»(“Driver”) επικαλείται παράλληλες λειτουργίες όπως οι “map,filter,reduce” σε ένα RDD σύνολο περνώντας μια συνάρτηση στο Spark, η οποία δρομολογεί την εκτέλεση της συνάρτησης παράλληλα στο υπολογιστικό σύμπλεγμα. Αυτές οι λειτουργίες και επιπρόσθετες όπως τα “joins” παίρνουν ένα RDD ως είσοδο και παράγουν ένα νέο RDD. Τα RDD είναι αμετάβλητες δομές δεδομένων και οι λειτουργίες τους είναι οκνηρές, η ανεκτικότητα στα σφάλματα επιτυγχάνεται μέσω της παρακολούθησης της γενεαλογίας κάθε RDD, της ακολουθίας των λειτουργιών που το παρήγαγαν, ώστε να μπορέσει να ανακατασκευαστεί σε περίπτωση απώλειας δεδομένων. Τα RDD μπορούν να περιέχουν οποιοδήποτε τύπο Python, Java ή Scala αντικειμένου.

Εκτός του RDD-oriented(RDD-προσανατολισμού) συναρτησιακού τύπου προγραμματισμού, το Spark παρέχει δύο κλειστού τύπου διαμοιραζόμενες μεταβλητές: μεταβλητές μετάδοσης (broadcast variables) οι οποίες αναφέρονται σε δεδομένα μόνο προς ανάγνωση όπου χρειάζονται να είναι διαθέσιμα σε όλους τους κόμβους και οι συσσωρευτές.



Σχήμα 15. Spark αρχιτεκτονική

Spark SQL

Το Spark SQL είναι ένα συστατικό στην κορυφή του Spark Core το οποίο εισάγει ένα νέο τύπο δεδομένων εν ονόματι DataFrames ο οποίος παρέχει υποστήριξη για δομημένα και ημιδομημένα δεδομένα. Παρέχει συγκεκριμένη γλώσσα για να διαχειριστεί τα DataFrames σε Scala, Java ή Python. Παρέχει επίσης υποστήριξη SQL γλώσσας με διεπαφές γραμμής εντολών και το διακομιστή ODBC / JDBC.

Spark Streaming

Το Spark Streaming αξιοποιεί την ικανότητα του Spark Core για δρομολόγηση για να εκτελέσει ανάλυση ροής δεδομένων (streaming analytics). Χωρίζει τα δεδομένα σε μίνι σωρούς και εκτελεί μετασχηματισμούς RDD σε αυτούς τους μικρούς σωρούς δεδομένων. Αυτός ο σχεδιασμός επιτρέπει το ίδιο σύνολο κώδικα που γράφτηκε για ανάλυση σωρού(batch analytics) να χρησιμοποιηθεί για ανάλυση ροών(streaming analytics).

MLlib Machine Learning Library

Το Spark Mlib αποτελεί ένα κατανεμημένο πλαίσιο μηχανικής μάθησης στην κορυφή του πυρήνα του Spark. Εξαιτίας του ότι μεγάλο μέρος της αρχιτεκτονικής του Spark είναι βασισμένη στην κατανεμημένη μνήμη είναι έως και 9 φορές γρηγορότερο απ' το βασισμένο στο δίσκο Apache Mahout. Πολλοί κοινοί αλγόριθμοι στατιστικής και μηχανικής μάθησης υλοποιήθηκαν στο MLlib οι οποίοι απλοποιούν μεγάλης κλίμακας αγωγούς μηχανικής μάθησης.

GraphX

Το GraphX είναι πλαίσιο κατανεμημένης επεξεργασίας γράφων στην κορυφή του Apache Core. Παρέχει ένα API για την έκφραση ενός υπολογισμού γράφου που μπορεί να μοντελοποιηθεί στην αφαίρεση Pregel και μάλιστα βελτιστοποιημένης εκτέλεσης.

Όπως το Apache Spark αρχικά ξεκίνησε ως ερευνητικό πρόγραμμα στο AMPLAB του US Berkley και στην Databricks[36] και αργότερα δόθηκε ως δωρεά στο Apache Software Foundation και στο πρόγραμμα Spark.

4.5 Εργαλεία προγραμματισμού που χρησιμοποιήθηκαν

Κάθε εργαλείο της πλατφόρμας που αναπτύχθηκε ξεχωριστά, υποστηρίζει μια ευρεία γκάμα από γλώσσες προγραμματισμού δίνοντας ευελιξία όσο αναφορά την επιλογή. Παρ' όλα αυτά κάποια εργαλεία δείχνουν να έχουν καλύτερη υποστήριξη σε συγκεκριμένη γλώσσα προγραμματισμού όπως το Spark για τη γλώσσα προγραμματισμού Scala.

Γλώσσες προγραμματισμού

Χρησιμοποιήθηκαν δύο γλώσσες προγραμματισμού οι οποίες είναι οι εξής:

- **Scala**

Η Scala αποτελεί μία γλώσσα προγραμματισμού γενικού σκοπού. Έχει πλήρη υποστήριξη συναρτησιακού προγραμματισμού και ένα πολύ ισχυρό σύστημα στατικού τύπου. Σχεδιασμένη για να είναι λακωνική, πολλές φορές σχεδιαστικές αποφάσεις της γλώσσας εμπνεύστηκαν απ' την κριτική των ελλείψεων της Java. Ο πηγαίος κώδικας μεταφράζεται σε Java bytecode έτσι ώστε ο εκτελέσιμος κώδικας που προκύπτει μπορεί να τρέχει σε μια εικονική μηχανή Java. Υποστηρίζει αρκετά χαρακτηριστικά συναρτησιακού προγραμματισμού.

- **Python**

Η Python είναι μια ευρέως χρησιμοποιούμενη υψηλού επιπέδου γενικού σκοπού, διερμηνευτή, δυναμική γλώσσα προγραμματισμού. Δημιουργήθηκε από τον Guido van Rossum το 1990 και η τελευταία έκδοση είναι η 3.5.1 στις 7 Δεκεμβρίου του 2015. Η σχεδιαστική της φιλοσοφία δίνει έμφαση στην ευκολία ανάγνωσης κώδικα καθώς η σύνταξη της επιτρέπει στους προγραμματιστές να γράφουν πολύ λιγότερο κώδικα απ' ότι η C++ ή η Java, είναι κατάλληλη για προγράμματα μικρής αλλά και μεγάλης κλίμακας καθώς και μεγάλη υποστήριξη σε βιβλιοθήκες. Υποστηρίζει πολλά παραδείγματα προγραμματισμού όπως αντικειμενοστραφή (object-oriented), προστακτικό (imperative), συναρτησιακό(functional) και διαδικαστικό(procedural).

Προγραμματιστικά πακέτα και βιβλιοθήκες

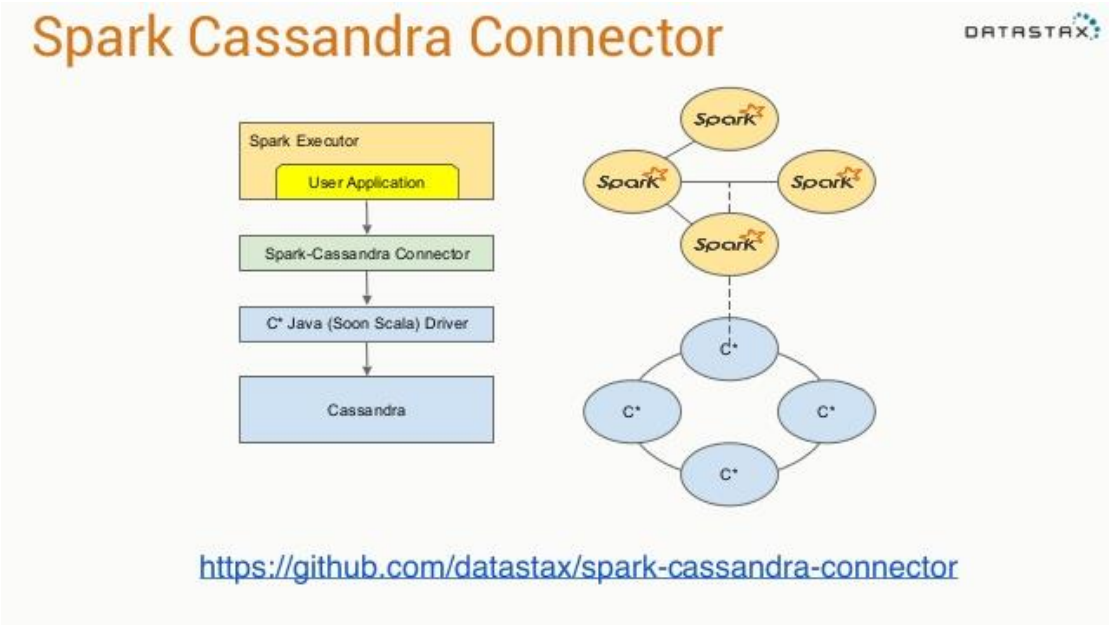
Για την προγραμματιστική επαφή με την Cassandra χρησιμοποιήθηκε ένας οδηγός πελάτη(client driver) ο DataStax Python Driver for Apache Cassandra[37]. Η τελευταία έκδοση είναι η 3.3.Ο οδηγός αυτός δουλεύει αποκλειστικά με την Cassandra Query Language v3 (CQL3) και με τα φυσικά πρωτόκολλά της Cassandra παρέχοντας υποστήριξη για εκδόσεις 1.2+. Για την Python υποστηρίζονται οι εκδόσεις 2.6, 2.7, 3.3, 3.4. Είναι ανοιχτού κώδικα κάτω απ' την άδεια της Apache.

Για την εκτέλεση προγραμμάτων στο Spark χρησιμοποιήθηκε το API της Apache για Scala [38].

Για την σύνδεση του Spark με την Cassandra χρησιμοποιήθηκε ο Spark Cassandra Connector της Datastax για Scala [39](ο οποίος είναι στην έκδοση 1.6.0 M2), επιτρέποντας την έκθεση των πινάκων(tables) της Cassandra σε Spark RDDs και το αντίστροφο, όπως επίσης και την εκτέλεση ερωτημάτων CQL στις εφαρμογές του Spark.

Επίσης εγκαταστάθηκαν η Java και χρησιμοποιήθηκε η βιβλιοθήκη Requests [40] για Python η οποία αποτελεί μια HTTP βιβλιοθήκη, ασφαλή και εύκολη σε χρήση για τον προγραμματιστή.

Για την προγραμματισμένη εκτέλεση των script χρησιμοποιήθηκε ο cron[41] , ένας job scheduler του Linux.



Σχήμα 16. Cassandra και Spark μέσω Connector

Επισκόπηση κεφαλαίου

Τα κύρια εργαλεία που χρησιμοποιήθηκαν είναι το Zabbix, η Apache Cassandra και το Apache Spark.Ο λόγοι που χρησιμοποιήθηκαν τα κύρια συγκεκριμένα εργαλεία:

Zabbix
Το Zabbix έχει Rest API μέσω του οποίου είναι δυνατή η άντληση των δεδομένων που παράγει μέσω προγραμμάτων ,είτε ιστορικών είτε δεδομένων που παράγονται ζωντανά. Επιτρέπει την εκτέλεση ρουτινών σε κάθε Agent. Επίσης εκτός των έτοιμων προφίλ δίνει τη δυνατότητα βαθιάς παραμετροποίησης και ευελιξίας.

Apache Cassandra & Apache Spark
Τα δύο αυτά εργαλεία αποτελούν τεχνολογίες αιχμής στους τομείς τους, είναι εργαλεία που αναπτύσσονται ραγδαία τα τελευταία χρόνια και οι επιδόσεις τους είναι από τις καλύτερες σε σύγκριση με άλλες λύσεις που κυκλοφορούν [42]. Αυτό σε συνδυασμό με τον Spark Cassandra Connector της Datastax ο οποίος διευκολύνει σε μεγάλο βαθμό την πρόσβαση του Spark.

Οι εκδόσεις των εργαλείων αυτών:
Python 3.4
Scala 2.10.4
Java 1.8.0_72-b15
ApacheSpark 1.5.2
Apache Cassandra 3.2
Zabbix 3.0
Spark Cassandra Connector 1.5.0
DataStax Python Driver for Apache Cassandra 3.0.0
Requests 2.9.1

Κεφάλαιο 5

Η αρχιτεκτονική της πλατφόρμας

Η υποδομή του εργαστηρίου χρησιμοποιεί το Openstack μία δωρεάν και ανοιχτού κώδικα πλατφόρμα για Υπηρεσίες Νέφους (Cloud Computing) παρέχοντας Υποδομή ως Υπηρεσία (Infrastructure as a Service). Η υποδομή αυτή χρησιμοποιείται απ' τα μέλη του εργαστηρίου κάνοντας χρήση εικονικών μηχανών (virtual machines) οι οποίες δημιουργούνται από το διαχειριστή (administrator) της υποδομής.

5.1 Η αρχιτεκτονική

Οι εικονικές μηχανές καθώς και οι φυσικές μηχανές του εργαστηρίου παρέχουν χρήσιμα δεδομένα για τον διαχειριστή της υποδομής τα οποία σκοπός της πλατφόρμας θα είναι να τα συλλέγει και να τα αποθηκεύει σε μια βάση δεδομένων για μετέπειτα επεξεργασία και παρουσίαση των αποτελεσμάτων στον διαχειριστή για την παρακολούθηση της υπηρεσίας και τη λήψη αποφάσεων.

Η αρχιτεκτονική της πλατφόρμας χωρίζεται στις εξής δομικές μονάδες:

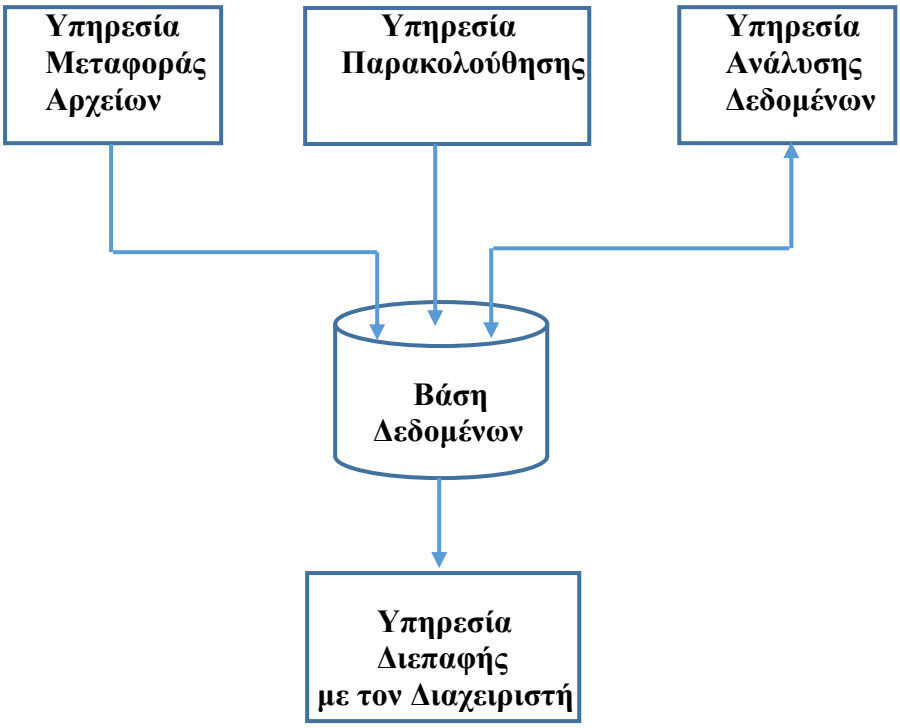
- **Υπηρεσία Παρακολούθησης**
  Η συγκεκριμένη υπηρεσία είναι υπεύθυνη για τη συνεχή συλλογή και προώθηση των δεδομένων των εικονικών μηχανών για αποθήκευση στη Βάση Δεδομένων.

- **Υπηρεσία Μεταφοράς Αρχείων**
  Η υπηρεσία αυτή είναι υπεύθυνη για την μεταφορά και αποθήκευση των log αρχείων που παράγουν οι υπηρεσίες του Openstack στη βάση.

- **Υπηρεσία Ανάλυσης Δεδομένων**
  Τα δεδομένα που θα παράγουν οι υπηρεσίες Παρακολούθησης και Μεταφοράς Αρχείων θα επεξεργάζονται, παρέχοντας χρήσιμη πληροφορία για τον διαχειριστή του Νέφους και θα αποθηκεύονται στη Βάση Δεδομένων.

- **Υπηρεσία Διεπαφής με τον Διαχειριστή**
  Είναι η μονάδα που εμφανίζει στο διαχειριστή χρήσιμη πληροφορία από τα αποτελέσματα της επεξεργασίας, οποιαδήποτε στιγμή ο διαχειριστής τα ζητήσει.

- **Βάση Δεδομένων**
  Η βάση δεδομένων αποτελεί το συνδετικό κρίκο όλων των παραπάνω μονάδων παρέχοντας αξιοπιστία τόσο στις εγγραφές όσο και στις αναγνώσεις του απαιτητικού όγκου δεδομένων της πλατφόρμας.



Σχήμα 17. Σχηματικό διάγραμμα αρχιτεκτονικής της πλατφόρμας

Βέλη που καταλήγουν στη βάση σημαίνουν εγγραφή και βέλη που φεύγουν σημαίνουν ανάγνωση.

5.2 Ανάλυση των υπηρεσιών

Υπηρεσία Παρακολούθησης

Σχήμα 18. Υπηρεσία Παρακολούθησης

Πρώτο συστατικό στοιχείο της μονάδας είναι το πρόγραμμα Zabbix το οποίο συλλέγει δεδομένα για τον επεξεργαστή, τη μνήμη, το δίσκο, το δίκτυο, εφαρμογές κ.α., από τις εικονικές μηχανές.

Συγκεκριμένα, σε κάθε εικονική μηχανή (VM) εγκαθίσταται ένας Zabbix Agent ο οποίος στέλνει τα δεδομένα στον Zabbix Server. Ο Zabbix Server συλλέγει τα στοιχεία που τoy στέλνουν οι Agents δίνοντας τη δυνατότητα εκμετάλλευσης τους.

Το επόμενο συστατικό κομμάτι της μονάδας είναι ο **Zabbix - Cassandra Connector** και αποτελείται από δύο υπομονάδες:

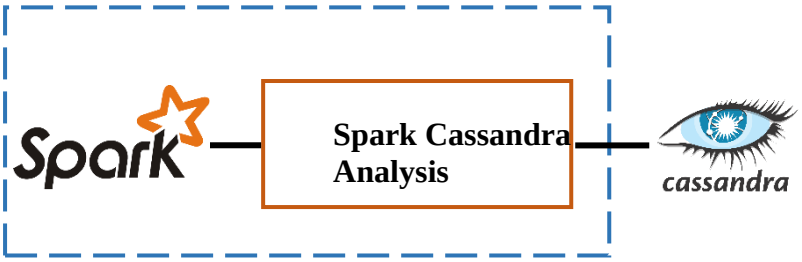
Η πρώτη αξιοποιώντας το Zabbix API, κάνει αιτήματα(requests) στον Zabbix Server και επιλέγει τα δεδομένα που θα συλλέξει η πλατφόρμα και η δεύτερη χρησιμοποιώντας τον connector της Python για Cassandra προωθεί τα δεδομένα αυτά όπου θα αποθηκευτούν στη βάση δεδομένων. Ουσιαστικά πρόκειται για μια σειρά από Python Scripts υπεύθυνα για την σύνδεση των δύο εργαλείων.

Υπηρεσία Μεταφοράς Αρχείων

Σχήμα 19. Υπηρεσία Μεταφοράς Αρχείων

Η Υπηρεσία Μεταφοράς Αρχείων αποτελείται από μια σειρά Python Scripts τα οποία έχουν πρόσβαση σε log αρχεία που παράγει η φυσική υποδομή του εργαστηρίου. Τα αρχεία αυτά αναλύονται και κάθε φορά που κάποια καινούργια πληροφορία δημιουργείται αποθηκεύεται σε πραγματικό χρόνο στη βάση δεδομένων με σκοπό την μετέπειτα επεξεργασία.

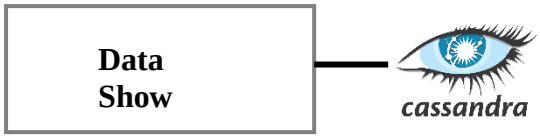
Υπηρεσία Ανάλυσης Δεδομένων



Σχήμα 20. Υπηρεσία Παρακολούθησης

Η Υπηρεσία Παρακολούθησης αποτελείται από το Spark το οποίο μέσω του Spark-Cassandra Connector της Datastax για Scala αποκτά πρόσβαση στην Cassandra για ανάγνωση και εγγραφή. Επεξεργάζεται τα δεδομένα που παράγει η εργαστηριακή υποδομή και που είναι αποθηκευμένα στη βάση δεδομένων και έπειτα αποθηκεύει τα αποτελέσματα πίσω στη βάση. Τη δουλειά αυτή αναλαμβάνει ο **Spark Cassandra Analysis** που δημιουργήθηκε αποτελούμενος ουσιαστικά από Scala Scripts,χάρη στο API του Connector της Datastax και του API του Spark, τα οποία αναλαμβάνουν την ανάγνωση των δεδομένων απ' την Cassandra,την επεξεργασία τόσο απευθείας στη βάση δεδομένων όσο και στο Spark Cluster αλλά και την εγγραφή των αποτελεσμάτων της επεξεργασίας αυτής στη βάση δεδομένων.

Υπηρεσία Διεπαφής με τον Διαχειριστή



Σχήμα 21. Υπηρεσία Διεπαφής με τον Διαχειριστή

Η Υπηρεσία Διεπαφής με τον Διαχειριστή αποτελείται από μια σειρά Python Scripts τα οποία εκτελώντας τα ο διαχειριστής αποκτά οποιαδήποτε στιγμή επιθυμεί πρόσβαση στα δεδομένα που παράγει η Υπηρεσία Ανάλυσης Δεδομένων.

Χάρη στην υπηρεσία αυτή ,συνδυάζοντας αποτελέσματα απ' τη φυσική υποδομή και τις εικονικές μηχανές, ο διαχειριστής της υποδομής μπορεί να δει χρήσιμα στοιχεία για την διαχείρισή της.

Κεφάλαιο 6

Πειραματική διαδικασία

6.1 Υλοποίηση της πλατφόρμας

Η υλοποίηση της πλατφόρμας έγινε στην υποδομή του εργαστηρίου. Το εργαστήριο περιλαμβάνει 11 Dell Servers Power Edge r220 κάθε ένας από τους οποίους διαθέτει επεξεργαστή Intel Xeon 4-core στα 3.2 GHz, μνήμη 8GB και δίσκο 1 TB. Η συνολική υποδομή δηλαδή διαθέτει 44 πυρήνες, 88 GB και 22 TB.

Για τις ανάγκες της πλατφόρμας χρησιμοποιήθηκαν 6 εικονικές μηχανές οι οποίες είναι συνδεδεμένες σε δίκτυο :

- spark1 - 10.0.2.7 με 2 core CPU, 4GB RAM και 40 GB disk
- spark2 - 10.0.2.8 με 2 core CPU, 4GB RAM και 40 GB disk
- cassandra1 - 10.0.2.9 με 2 core CPU, 4GB RAM και 40 GB disk
- cassandra2 - 10.0.2.10 με 2 core CPU, 4GB RAM και 120 GB disk
- zabbix - 10.0.2.11 με 1 core CPU, 4GB RAM και 40 GB disk
- dataservice - 10.0.2.12 με 1 core CPU, 4GB RAM και 120 GB disk

Σε όλες τις εικονικές μηχανές είναι εγκατεστημένος ο Zabbix Agent.

Τα spark1, spark2 αποτελούν το spark cluster με εγκατεστημένα τα Java,Scala,Spark,Spark Cassandra Connector και τον Spark Cassandra Analysis που δημιουργήθηκε.

Τα cassandra1, cassandra2 αποτελούν τους servers του cassandra cluster με εγκατεστημένα τη Java και την Cassandra.

Στον zabbix εγκαταστάθηκε ο Zabbix Server.

Τέλος στον dataservice είναι εγκατεστημένα, η βιβλιοθήκη Requests για Python, ο DataStax Python Driver for Apache Cassandra και οι Zabbix - Cassandra Connector,Log Cassandra Connector και Data Show που δημιουργήθηκαν.

6.1.1 Zabbix

Όπως αναφέρθηκε και παραπάνω σε κάθε μια από τις παραπάνω εικονικές μηχανές εγκαταστάθηκε από ένας Zabbix Agent, ο Zabbix Server εγκαταστάθηκε στο VM zabbix.

Στην παραμετροποίηση(configuration) του Zabbix δημιουργήθηκαν οι παρακάτω Hosts:

NAME ▲	APPLICATIONS	ITEMS	TRIGGERS	GRAPHS	DISCOVERY	WEB	INTERFACE	TEMPLATES	STATUS
Cassandra-1	Applications 12	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	10.0.2.9: 10050		Enabled
Cassandra-2	Applications 11	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	10.0.2.10: 10050		Enabled
data-service	Applications 11	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	10.0.2.12: 10050		Enabled
Spark-1	Applications 11	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	10.0.2.7: 10050		Enabled
Spark-2	Applications 11	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	10.0.2.8: 10050		Enabled
zabbix-server	Applications 11	Items 70	Triggers 43	Graphs 12	Discovery 2	Web	127.0.0.1: 10050		Enabled

Σχήμα 22. Zabbix εξυπηρετητές

Έπειτα δημιουργήθηκε μια ομάδα (group) για λόγους οργάνωσης αλλά και για λόγους ευκολίας όπως θα αναλυθεί παρακάτω. Το group που δημιουργήθηκε ονομάζεται "Thesis servers" όπως φαίνεται στην παρακάτω εικόνα.

Thesis servers	Hosts 6	Templates	Cassandra-1, Cassandra-2, data-service, Spark-1, Spark-2, zabbix-server

Σχήμα 23. Zabbix ομάδα εξυπηρετητών

Τα δεδομένα τα οποία θέλουμε να συλλέξουμε για την κάθε εικονική μηχανή χωρίζονται σε τρεις κατηγορίες:

- Για τον επεξεργαστή, τη μνήμη, το δίκτυο και το δίσκο του συστήματος
- Για σκανδάλες(triggers)
- Για τους πόρους που καταναλώνουν οι εφαρμογές

Κάθε εξυπηρετητής όπως φαίνεται και παραπάνω περιλαμβάνει Εφαρμογές (Applications). Τα Applications χρησιμοποιούνται για την ομαδοποίηση των Αντικειμένων (Items) τα οποία θα εισάγουμε στη βάση δεδομένων.

Για τον κάθε εξυπηρετητή τα Applications τα οποία επιλέχθηκαν είναι τα εξής:

APPLICATION ▲	ITEMS
CPU	Items 13
Filesystems	Items 5
Memory	Items 5
Network interfaces	Items 2

Σχήμα 24. Zabbix εφαρμογές

Τα Applications προέρχονται απ' το έτοιμο πρότυπο (template) που υπήρχε στο Zabbix, το Template OS Linux .Είναι αναγκαίο να αναφερθεί ότι για να επιτευχθεί αλλαγή στα αντικείμενα οποιουδήποτε template είναι απαραίτητο μετά την εφαρμογή του να γίνει αποσύνδεση (unlink) μέσω παραμετροποίησης του host. Έτσι τα Items μένουν αλλά πλέον είναι εφικτή η εφαρμογή αλλαγών.

Τα Items τα οποία επιλέχθηκαν για εισαγωγή στη βάση είναι τα εξής:

- Για τον επεξεργαστή

WIZARD	NAME	TRIGGERS	KEY ▲	INTERVAL
	Interrupts per second		system.cpu.intr	5s
	Processor load (1 min average per core)	Triggers 1	system.cpu.load[percpu,avg1]	5s
	Processor load (5 min average per core)		system.cpu.load[percpu,avg5]	5s
	Processor load (15 min average per core)		system.cpu.load[percpu,avg15]	5s
	Context switches per second		system.cpu.switches	5s
	CPU idle time		system.cpu.util[,idle]	5s
	CPU interrupt time		system.cpu.util[,interrupt]	5s
	CPU iowait time	Triggers 1	system.cpu.util[,iowait]	5s
	CPU nice time		system.cpu.util[,nice]	5s
	CPU softirq time		system.cpu.util[,softirq]	5s
	CPU steal time		system.cpu.util[,steal]	5s
	CPU system time		system.cpu.util[,system]	5s
	CPU user time		system.cpu.util[,user]	5s

Σχήμα 25. Zabbix αντικείμενα εφαρμογής για τον επεξεργαστή

- Για τη μνήμη

NAME	TRIGGERS	KEY ▲	INTERVAL
Free swap space		system.swap.size[,free]	5s
Free swap space in %	Triggers 1	system.swap.size[,pfree]	5s
Total swap space		system.swap.size[,total]	5s
Available memory	Triggers 1	vm.memory.size[available]	5s
Total memory		vm.memory.size[total]	5s

Σχήμα 26. Zabbix αντικείμενα εφαρμογής για τη μνήμη

- Για το δίσκο

NAME	TRIGGERS	KEY ▲	INTERVAL
Mounted filesystem discovery: Free inodes on / (percentage)	Triggers 1	vfs.fs.inode[/,pfree]	5s
Mounted filesystem discovery: Free disk space on /		vfs.fs.size[/,free]	5s
Mounted filesystem discovery: Free disk space on / (percentage)	Triggers 1	vfs.fs.size[/,pfree]	5s
Mounted filesystem discovery: Total disk space on /		vfs.fs.size[/,total]	5s
Mounted filesystem discovery: Used disk space on /		vfs.fs.size[/,used]	5s

Σχήμα 27. Zabbix αντικείμενα εφαρμογής για τον δίσκο

- Για το δίκτυο

NAME	TRIGGERS	KEY ▲	INTERVAL
Network interface discovery: Incoming network traffic on eth0		net.if.in[eth0]	5s
Network interface discovery: Outgoing network traffic on eth0		net.if.out[eth0]	5s

Σχήμα 28. Zabbix αντικείμενα εφαρμογής για το δίκτυο

Το διάστημα 5 δευτερολέπτων (INTERVAL 5s) είναι παραμετροποίηση που έγινε σε όλους τους εξυπηρετητές και για όλα τα αντικείμενα καθώς τα scripts που δημιουργήθηκαν για να έχουν πρόσβαση στα παραπάνω αντικείμενα είναι σχεδιασμένα ώστε να λαμβάνουν τιμές κάθε 5 δευτερόλεπτα ,όπως θα φανεί και παρακάτω. Αν δεν άλλαζε το διάστημα, σε κάθε επανάληψη τα scripts θα εισήγαγαν στη βάση τα ίδια δεδομένα πολλαπλές φορές το οποίο είναι ανεπιθύμητο.

Οι σκανδάλες που χρησιμοποιήθηκαν:

Warning	/etc/passwd has been changed on {HOST.NAME}
Information	Configured max number of opened files is too low on {HOST.NAME}
Information	Configured max number of processes is too low on {HOST.NAME}
Warning	Disk I/O is overloaded on {HOST.NAME}
Warning	Mounted filesystem discovery: Free disk space is less than 20% on volume /
Warning	Mounted filesystem discovery: Free inodes is less than 20% on volume /
Information	Host information was changed on {HOST.NAME}
Information	Host name of zabbix_agentd was changed on {HOST.NAME}
Information	Hostname was changed on {HOST.NAME}
Average	Lack of available memory on server {HOST.NAME}
Warning	Lack of free swap space on {HOST.NAME}
Average	Less than 5% free in the value cache
Average	Less than 25% free in the configuration cache
Average	Less than 25% free in the history cache
Average	Less than 25% free in the text history cache
Average	Less than 25% free in the trends cache
Average	Less than 25% free in the vmware cache
Warning	More than 100 items having missing data for more than 10 minutes
Warning	Processor load is too high on {HOST.NAME}
Warning	Too many processes on {HOST.NAME}
Warning	Too many processes running on {HOST.NAME}
Information	Version of zabbix_agent(d) was changed on {HOST.NAME}
Average	Zabbix agent on {HOST.NAME} is unreachable for 5 minutes
Average	Zabbix alerter processes more than 75% busy

Σχήμα 29. Zabbix Σκανδάλες α)

Average	Zabbix configuration syncer processes more than 75% busy
Average	Zabbix db watchdog processes more than 75% busy
Average	Zabbix discoverer processes more than 75% busy
Average	Zabbix escalator processes more than 75% busy
Average	Zabbix history syncer processes more than 75% busy
Average	Zabbix housekeeper processes more than 75% busy
Average	Zabbix http poller processes more than 75% busy
Average	Zabbix icmp pinger processes more than 75% busy
Average	Zabbix ipmi poller processes more than 75% busy
Average	Zabbix java poller processes more than 75% busy
Average	Zabbix poller processes more than 75% busy
Average	Zabbix proxy poller processes more than 75% busy
Average	Zabbix self-monitoring processes more than 75% busy
Average	Zabbix snmp trapper processes more than 75% busy
Average	Zabbix timer processes more than 75% busy
Average	Zabbix trapper processes more than 75% busy
Average	Zabbix unreachable poller processes more than 75% busy
Average	Zabbix vmware collector processes more than 75% busy
Information	{HOST.NAME} has just been restarted

Σχήμα 30. Zabbix Σκανδάλες β)

Οι σκανδάλες προέρχονται απ' το template του Zabbix "Template OS Linux".

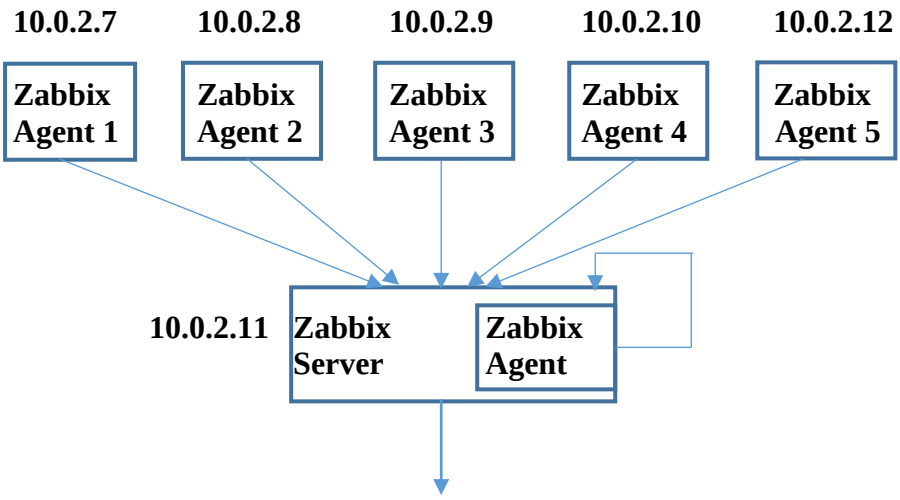
Για τη λήψη δεδομένων για τις εφαρμογές που τρέχουν σε κάθε εξυπηρετητή ήταν απαραίτητη η δημιουργία ενός script. Αυτό έγινε μέσω του frontend του Zabbix. Το script φαίνεται στην παρακάτω εικόνα:

Scripts

NAME ▲	TYPE	EXECUTE ON	COMMANDS
CPU-RAM UTIL PER PROCESS	Script	Agent	visudo ps aux

Σχήμα 31. Zabbix script

Χάρη στην εντολή ps aux που μπορεί να τρέξει σε κάθε agent καθίσταται δυνατή η συλλογή στοιχείων για τη μνήμη και τον επεξεργαστή που καταναλώνει κάθε εφαρμογή που τρέχει. Η εντολή visudo χρησιμοποιείται για εμφάνιση αποτελεσμάτων συμπεριλαμβανομένου και του root.



Σχήμα 32. Σύμπλεγμα Zabbix

6.1.2 Zabbix - Cassandra Connector

Ο Zabbix - Cassandra Connector αποτελείται από τρία python script. Το Zabbix_Script1 είναι υπεύθυνο για την διαχείριση των Items που περιλαμβάνονται στα Applications Memory, CPU, FileSystems και Network Interface. Το Zabbix_Script2 είναι υπεύθυνο για τη διαχείριση των σκανδαλών και το Zabbix_Script3 για τη διαχείριση των δεδομένων που αφορούν τους πόρους σε επεξεργαστική ισχύ και μνήμη που καταναλώνουν οι εφαρμογές που τρέχει το κάθε VM.

Κοινό και για τις τρεις ρουτίνες είναι η πιστοποίηση. Κάνοντας login χρησιμοποιώντας το όνομα χρήστη και τον κωδικό πρόσβασης ,όπως παρουσιάζεται παρακάτω, αποκτάμε ένα νόμισμα πιστοποίησης (authentication token) και μπορούμε να έχουμε πρόσβαση στον Zabbix Server.

```
url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
headers = {"Content-Type":"application/json-rpc"}
user = 'Admin'
password = 'zabbix'
login={
      "jsonrpc": "2.0",
      "method": "user.login",
      "params": {
            "user": user,
            "password": password
      },
      "id": 1,
      "auth": None
}
r = requests.post(url,headers=headers,json=login)
self.auth =(r.json())['result']
```

- **Zabbix_Script1 – *read_zabbix.py***

Η πρώτη ρουτίνα είναι υπεύθυνη για τη λήψη των πιο πρόσφατων τιμών των Items, που ανήκουν στα Applications “Memory, CPU, FileSystems και Network Interface”, απ’ τον Zabbix Server. Αυτό κάνει το πρώτο κομμάτι της ρουτίνας.

```
group="Thesis servers"
memory = {          ###memory
                  "jsonrpc": "2.0",
                  "method": "item.get",
                  "params": {
                    "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                    "group": group,
                    "application" : "Memory",
                  },
                  "auth": auth,
                  "id": 1
              }

cpu = {             ###cpu
                  "jsonrpc": "2.0",
                  "method": "item.get",
                  "params": {
                    "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                    "group": group,
                    "application" : "CPU",
                  },
                  "auth": auth,
                  "id": 1
              }

disk = {            ###disk
                      "jsonrpc": "2.0",
                      "method": "item.get",
                      "params": {
                        "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                        "group": group,
                        "application" : "Filesystems",
                      },
                      "auth": auth,
                      "id": 1
                  }

net = {             ###network
                      "jsonrpc": "2.0",
                      "method": "item.get",
                      "params": {
                        "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                        "group": group,
                        "application" : "Network interfaces",
                      },
                      "auth": auth,
                      "id": 1
                  }
```

```
#get cpu data
r = requests.post(url,headers=headers,json=cpu)
x=r.json()['result']

#get ram data
r = requests.post(url,headers=headers,json=memory)
y=r.json()['result']

#get disk data
r = requests.post(url,headers=headers,json=disk)
z=r.json()['result']

#get network data
r = requests.post(url,headers=headers,json=net)
w=r.json()['result']
```

Χρησιμοποιώντας τη μέθοδο item.get του zabbix api μπορούμε να πάρουμε τα δεδομένα ακριβώς που θέλουμε.

Μέσω της χρήσης της παραμέτρου “output” λαμβάνουμε τα στοιχεία που θέλουμε για τα αντικείμενα μας, συγκεκριμένα το όνομα του αντικειμένου “name”, η τιμή “lastvalue”, το κλειδί του αντικειμένου “key_”, η στιγμή που έγινε η μέτρηση“lastclock”, ο κωδικός του αντικειμένου “itemid” και του οικοδεσπότη “hostid”.

Μέσω της παραμέτρου “application” επιλέγουμε τα αντικείμενα που θέλουμε.

Χρησιμοποιώντας την παράμετρο “group” κατά την μελλοντική πρόσθεση κι άλλων εξυπηρετητών τοποθετώντας τους στο ίδιο group δεν χρειάζεται διακοπή της ρουτίνας καθώς αυτόματα προστίθεται ο νέος εξυπηρετητής και τα δεδομένα ρέουν στη βάση.

Το δεύτερο κομμάτι της ρουτίνας είναι υπεύθυνο για την αποθήκευση των δεδομένων αυτών στην Cassandra.

```
result=x+y+z+w

cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()

insert_statement  = session.prepare("""
      INSERT INTO spark.z (uid,timestamp,name,key, value,
itemid,hostid)
      VALUES (?,?,?,?,?,?,?);
      """)

for w in result:
      session.execute(insert_statement,[uuid.uuid1(),int(w['lastclock
']),w['name'],w['key_'],w['lastvalue'],int(w['itemid']),int(w["hostid
"])])

session.cluster.shutdown()
```

Τα αποτελέσματα του προηγούμενου τμήματος της ρουτίνας συνοψίζονται για να περαστούν στη βάση σε ένα ενιαίο ερώτημα(query). Σημαντικό κομμάτι του κώδικα είναι οι έτοιμες δηλώσεις (prepared statements) μέσω των οποίων υπάρχουν οφέλη

στην απόδοση καθώς εκτελείται το ίδιο query έστω και με διαφορετικές παραμέτρους. Εκτός των άλλων δεδομένων εισάγεται μαζί με το κάθε αντικείμενο και ένα ξεχωριστό-μοναδικό κλειδί για κάθε στοιχείο της βάσης, το uuid ώστε να ξεχωρίζει απ' τα άλλα σε περίπτωση εισαγωγής της ίδιου ακριβώς στοιχείου.
Ο παραπάνω κώδικας στην πλήρη του μορφή και με τροποποιήσεις τρέχει συνεχόμενα στον server dataservice και κάθε 5 δευτερόλεπτα προωθεί τα δεδομένα και τα αποθηκεύει στην Cassandra.

Ο πλήρης κώδικας της ρουτίνας όπως και όλων των υπολοίπων παρουσιάζεται στο Παράρτημα Β.

- Zabbix_Script2 – *zabbix_triggers.py*

Η ρουτίνα αυτή είναι υπεύθυνη για τη συλλογή πληροφοριών για τις σκανδάλες του κάθε εξυπηρετητή.

```
cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()
url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
headers = {"Content-Type":"application/json-rpc"}
group="Thesis servers"

insert_statement  = session.prepare("""
      INSERT INTO spark.z_triggers2
(uid,timestamp,triggerid,description,priority,lastchange,hostid,value
)
      VALUES (?,?,?,?,?,?,?,?);
      """)

trigger = {
      "jsonrpc": "2.0",
      "method": "trigger.get",
      "params": {
            "output": [
            "triggerid",
            "description",
            "priority",
                  "lastchange",
                  "value",
        ],
            "group": group,
      },
      "auth": auth,
    "id": 1,
}
r = requests.post(url,headers=headers,json=trigger)
result=r.json()['result']
taim=time.time()
for w in result:
      host = {
            "jsonrpc": "2.0",
            "method": "host.get",
            "params": {
                        "output": "hostid",
                        "triggerids" : w['triggerid'],
        },
        "auth": auth,
        "id": 1,
```

```
    }
      r = requests.post(url,headers=headers,json=host)
      hostid=r.json()['result'][0]['hostid']
      session.execute(insert_statement,[uuid.uuid1(),taim,int(w['trig
gerid']),w['description'],int(w['priority']),int(w['lastchange']),int
(hostid)],int(w['value']))

session.cluster.shutdown()
```

Τα δεδομένα που θέλουμε να περάσουμε για την κάθε σκανδάλη είναι ξεχωριστό uuid, η χρονική στιγμή που ελήφθη η μέτρηση, το id της trigger,περιγραφή για ανθρώπινη ανάγνωση, προτεραιότητα με την έννοια της σημαντικότητας της σκανδάλης και το id του εξυπηρετητή. Χρησιμοποιώντας την trigger.get του zabbix api λαμβάνουμε όλα τα δεδομένα που χρειαζόμαστε εκτός απ' το hostid καθώς η συγκεκριμένη μέθοδος δεν το υποστηρίζει στην παρούσα έξοδο γι' αυτό χρησιμοποιήθηκε και η μέθοδος host.get όπου χρησιμοποιώντας το triggerid αποκτάμε και τα hostid που χρειαζόμαστε για να ξεχωρίζουμε το τί ανήκει σε ποιον. Οι σκανδάλες που συλλέγονται είναι αυτές που είναι ενεργοποιημένες μέσω του φίλτρου value στην trigger.get.

Το τελευταίο κομμάτι είναι το πέρασμα αυτών των δεδομένων στη βάση.

- Zabbix_Script3 – *zabbix_script.py*

Η τρίτη ρουτίνα του Zabbix - Cassandra Connector είναι υπεύθυνη για την εκτέλεση του zabbix script ps aux στους zabbix agents του group και μέσω αυτού να καθίσταται δυνατή η αποθήκευση δεδομένων που αφορούν την επεξεργαστική ισχύ και τη μνήμη που καταναλώνει η κάθε εφαρμογή στον κάθε εξυπηρετητή.

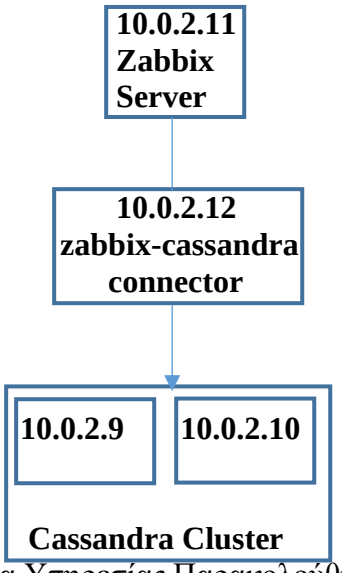
```
cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()

url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
headers = {"Content-Type":"application/json-rpc"}

insert_statement  = session.prepare("""
        INSERT INTO spark.z_proccesses
(uid,timestamp,hostid,user,pid,cpu,ram,starts,lasts,command)
        VALUES (?,?,?,?,?,?,?,?,?,?);
        """)

groupid = 8

host = {
      "jsonrpc": "2.0",
      "method": "host.get",
            "params": {
                  "output": ["hostid","host"],
                  "groupids" : groupid,
            },
            "auth": auth,
            "id": 1,
      }
r = requests.post(url,headers=headers,json=host)
hostids=r.json()['result']
for i in hostids:
      script_exec = {
            "jsonrpc": "2.0",
```

```
            "method": "script.execute",
            "params": {
                        "hostid" : i['hostid'],
                        "scriptid" : 5,
            },
            "auth": auth,
            "id": 1
      }
      r = requests.post(url,headers=headers,json=script_exec)
      script=r.json()['result']['value']

      script2=str(script).split('\n')

      for j in range(2,len(script2)):
            x=script2[j].split()
            c= x[10:]
            command = ' '.join(c)

      session.execute(insert_statement,[uuid.uuid1(),int(time.time())
,int(i['hostid']),x[0],int(x[1]),float(x[2]),float(x[3]),x[8],x[9],co
mmand])
session.cluster.shutdown()
```

Οι παραπάνω ρουτίνα χρησιμοποιεί τη μέθοδο script.execute για την εκτέλεση του ps aux script. Χρησιμοποιώντας τη μέθοδο host.get περιορίζουμε την εκτέλεση της ρουτίνας μόνο για τους εξυπηρετητές του group. Το τελευταίο κομμάτι είναι υπεύθυνο για την προτασιακή ανάλυση και το πέρασμα των δεδομένων στη βάση.

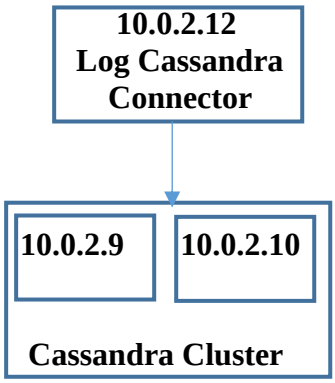
Όπως έχει αναφερθεί και παραπάνω όλα τα script τρέχουν κάθε 5 δευτερόλεπτα ώστε να εξασφαλίζεται η λεπτομερής καταγραφή των πληροφοριών. Σε ψευδοκώδικα για python αυτό γίνεται:

```
while True:
      start_time = time.time()              #start time of execution
      zabbix_script.execution               #this line is pseudocode

      exec_time = time.time()-start_time
      time.sleep(abs(5-exec_time))          #end time of execution
```



Σχήμα 33. Σύμπλεγμα Υπηρεσίας Παρακολούθησης

6.1.3 Log Cassandra Connector



Σχήμα 34. Σύμπλεγμα Υπηρεσίας Μεταφοράς Αρχείων

Ο Log Cassandra Connector αποτελείται από μια σειρά ρουτινών οι οποίες είναι υπεύθυνες για τη μεταφορά των δεδομένων από τα log αρχεία ,που παράγουν οι υπηρεσίες του Openstack, στη βάση δεδομένων. Τόσο ο Log Cassandra Connector όσο και τα log αρχεία βρίσκονται στον host 10.0.2.12 .

Πιο συγκεκριμένα οι υπηρεσίες από τις οποίες συλλέγουμε πληροφορίες είναι οι glance_controller, neutron_compute1, neutron_controller και nova_controller.

Τα αρχεία των οποίων τα δεδομένα περνάμε στη βάση είναι τα εξής:

- Από glance_controller

glance-api.log.1
glance-registry.log.1

- Από neutron_compute1

neutron-linuxbridge-agent.log.1

- Από neutron_controller

dhcp-agent.log.1
l3-agent.log.1
nova-consoleauth.log.1
nova-novncproxy.log.1
nova-scheduler.log

- Από nova_controller

nova-api.log.1
nova-consoleauth.log.1

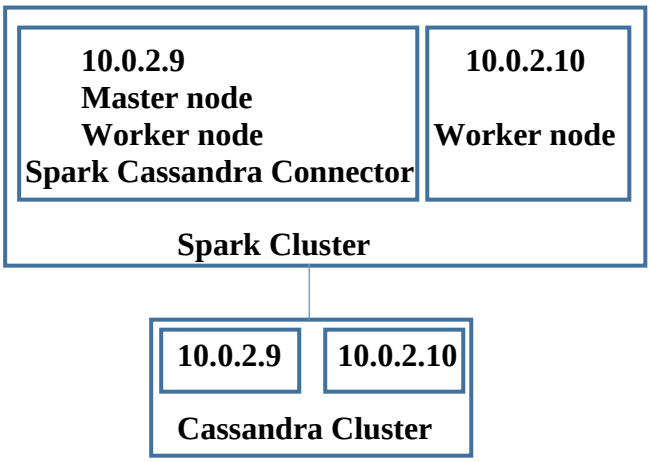
Έχουμε φτιάξει από μια ρουτίνα για κάθε αρχείο. Μελετώντας τα αρχεία, οι ρουτίνες ομαδοποιούνται σε δύο κατηγορίες όσον αφορά τη δομή των log που διαβάζουν. Λόγω του ότι οι ρουτίνες όλες έχουν παρόμοια δομή θα αναλυθεί μια από αυτές. Σε πλήρη μορφή όπως και οι υπόλοιπες ρουτίνες που είναι αναγκαίες για την πλατφόρμα μας θα εμφανιστούν στο Παράρτημα Β.

```
filename = 'Server_Logs/nova_controller/nova-api.log.1'
x=1    #old lines
y=1    #new lines
cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()
insert_statement  = session.prepare("""
      INSERT INTO spark.logs_1
(uid,date,time,id,priority,service,host,info)
      VALUES (?,?,?,?,?,?,?,?);
        """)

while True:
      try:
            with open(filename, 'r') as f:
                  x=y
                  y=sum(1 for _ in f)+1
            if y<x:
                  x=1

            for i in range(x,y):
                  l=linecache.getline(filename,i).rstrip()
                  l2=l.split()

      session.execute(insert_statement,[uuid.uuid1(),l2[0],
                        l2[1],l2[2],l2[3],l2[4],l2[6],"
".join(l2[7:])])
            linecache.clearcache()
            time.sleep(10)

      except KeyboardInterrupt:
            print('Session closed')
            session.cluster.shutdown()
            sys.exit()
```

Η παραπάνω ρουτίνα είναι υπεύθυνη για το αρχείο “nova-api.log.1”. Κάθε 10 δευτερόλεπτα ελέγχει αν προστέθηκε νέα πληροφορία στο αρχείο και περνάει τα νέα δεδομένα στη βάση. Λόγω της φύσης των log αρχείων να ξεκινάνε απ’ την αρχή αφού φτάσουν ένα συγκεκριμένο μέγεθος (και αποθηκεύουν συμπιεσμένο το αρχείο) ήταν απαραίτητος ένας επιπλέον έλεγχος για την λειτουργία της ρουτίνας.

6.1.4 Spark & Spark Cassandra Analysis

Έχοντας τα δεδομένα που θέλουμε στη βάση επόμενο στάδιο είναι η ανάλυση τους. Οι servers 10.0.2.7, 10.0.2.8 αποτελούν το spark cluster. Και στους δύο server είναι εγκατεστημένοι οι worker nodes που είναι υπεύθυνοι για την επεξεργασία. Ο master node που διαχειρίζεται το cluster είναι εγκατεστημένος στον 10.0.2.7 όπως επίσης και ο Spark Cassandra Connector που είναι υπεύθυνος για την πρόσβαση του Spark στην Cassandra. Όλα αυτά παρουσιάζονται στο παρακάτω σχήμα:



Σχήμα 35. Σύμπλεγμα Υπηρεσίας Ανάλυσης Δεδομένων

Ο Spark Cassandra Connector αποτελείται από οκτώ scala προγράμματα τα οποία εκτελούνται απ' τον master node. Τα πρώτα επτά επεξεργάζονται δεδομένα των εικονικών μηχανών και το όγδοο των φυσικών μηχανών της υποδομής.

Τα κύρια μέρη των προγραμμάτων αυτών είναι τα εξής:

- **Scala Script 1**

Το πρώτο πρόγραμμα είναι υπεύθυνο για την εξαγωγή του μέσου όρου των τιμών του ποσοστού χρησιμοποίησης του επεξεργαστή απ' το χρήστη, της διαθέσιμης μνήμης, του δικτύου και του δίσκου του κάθε εξυπηρετητή. Οι πρώτες δύο μέθοδοι "where" και "select" αντίστοιχα είναι μέθοδοι που εκτελούνται στη βάση απευθείας. Στην δική μας αρχιτεκτονική όπου το Spark Cluster και το Cassandra Cluster είναι σε διαφορετικούς servers είναι καλύτερο από άποψη μείωσης της κίνησης του δικτύου να γίνεται το βασικό φιλτράρισμα των δεδομένων απευθείας στη βάση και όχι μεταφορά των δεδομένων και επεξεργασία απευθείας απ' το Spark Cluster.

Επειδή η where δεν επιτρέπει την επιλογή πολλαπλών κλειδιών της ίδιας στήλης στην υπάρχουσα έκδοση του API χρησιμοποιούμε την "filter" που ανήκει στο Spark API. Έπειτα χρησιμοποιώντας την τεχνική MapReduce βρίσκουμε τους μέσους όρους των δεδομένων. Τέλος αποθηκεύουμε τα δεδομένα αυτά στη βάση με τη μέθοδο saveToCassandra. Πριν την αποθήκευση των δεδομένων ο πίνακας που προέκυψε απ' την collect μετατράπηκε σε λίστα με έξτρα πληροφορίες για το table των αποτελεσμάτων.

```
    val conf = new
SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_1",
conf)
    val rdd= sc.cassandraTable("spark", "z")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24
```

```
    val rdd2=
rdd.select("hostid","value","itemid","key").where("timestamp>=? and
timestamp<=?",myfilter,current_time).filter(x=>x.getString("key")=="v
m.memory.size[available]" || x.getString("key")==
"system.cpu.util[,user]" || x.getString("key")=="net.if.in[eth0]" ||
x.getString("key")=="net.if.out[eth0]" || x.getString("key")==
"vfs.fs.size[/,used]")
    val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getInt("itemid"),x.getString("key"
)),(x.getFloat("value"),1))).reduceByKey((x,y)=>(x._1 + y._1, x._2 +
y._2)).collect

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,Int,String,Float)]()
    for (i<- z){
      myList  =
(uuid,current_time.toInt,i._1._1,i._1._3,i._2._1/i._2._2) :: myList
     }
     val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats1",SomeColumns
("uid"," timestamp","hostid","key","value"))
```

- **Scala Script 2**

Το δεύτερο πρόγραμμα είναι υπεύθυνο για την εξαγωγή του ποσοστού που ο επεξεργαστής ξεπέρασε ένα κατώφλι(στην παρούσα έκδοση την τιμή 30).Το χρονικό διάστημα μεταβάλλεται όπως και στο Scala Script 2 όπως επιθυμεί ο διαχειριστής. Στην παρούσα έκδοση τα δεδομένα περιορίζονται στη μια ημέρα.

```
val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_2",
conf)
    val rdd= sc.cassandraTable("spark", "z")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24
    val lim : Float = 30

    val rdd2= rdd.select("hostid","itemid","value",
"key").where("timestamp>=? and
timestamp<=?",myfilter,current_time).filter(x=> x.getString("key")==
"system.cpu.util[,user]")

    val z  =
rdd2.map(x=>(x.getInt("hostid"),(if(x.getFloat("value")>=lim) 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,Int,Float,Float)]()
    for (i<- z){
      myList  =
(uuid,current_time.toInt,i._1,lim,i._2._1.toFloat/i._2._2.toFloat) ::
myList
    }
```

```
    val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats2",SomeColumns
("uid"," timestamp","hostid","lim","value"))
```

- **Scala Script 3**

Το Scala Script 3 αποθηκεύει στην Cassandra τις ενεργοποιημένες σκανδάλες καθώς και το ποσοστό της διάρκειας που ήταν ενεργοποιημένες σε σχέση με τη διάρκεια που ήταν απενεργοποιημένες σε διάστημα μιας ημέρας για όλες τις εικονικές μηχανές.

```
    val conf = new
SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_3",
conf)
    val rdd= sc.cassandraTable("spark", "z_triggers2")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24

    val rdd2= rdd.where("timestamp>=? and
timestamp<=?",myfilter,current_time).select("triggerid","description"
,"priority","lastchange","hostid","value")

   val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("description"),x.getInt(
"priority"),x.getInt("lastchange"),x.getInt("triggerid")),(if(x.getFl
oat("value")==1) 1 else 0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 +
y._2)).collect

   def uuid = java.util.UUID.randomUUID

   var myList =
List[(java.util.UUID,Int,Int,String,Int,Int,Int,Float)]()
   for (i<- z){
     myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,i._1._4,i._1._5,i._2
._1.toFloat/i._2._2) :: myList
   }
   val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats3",SomeColumns
("uid","
timestamp","hostid","description","priority","lastchange","triggerid"
,"value"))
```

- **Scala Script 4**

.

```
    val conf = new
SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_4",
conf)
    val rdd= sc.cassandraTable("spark", "z_proccesses")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
```

```
    val myfilter = current_time-n*3600*24

    val rdd2=
rdd.where("cpu>?",0.1).select("hostid","user","cpu","command").where(
"timestamp>=? and timestamp<=?",myfilter,current_time)

    val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("com
mand")),(x.getFloat("cpu"),1))).reduceByKey((x,y)=>(x._1 + y._1,x._2
+ y._2)).collect

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,Int,String,String,Float)]()
    for (i<- z){
      myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,i._2._1/i._2._2) ::
myList
    }
    val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats4",SomeColumns
("uid"," timestamp","hostid","user","command","value"))
```

Το Scala Script 4 αποθηκεύει στην Cassandra για κάθε εξυπηρετητή το μέσο όρο των πόρων που καταναλώνει η κάθε διεργασία όλων των χρηστών σε μία ημέρα. Χρησιμοποιείται το φιλτράρισμα “cpu>0.1” για την αποφυγή καταγραφής μετρήσεων για διεργασίες οι οποίες καταναλώνουν ελάχιστους πόρους.

- **Scala Script 5**

Το Scala Script 5 αποθηκεύει στην Cassandra για κάθε εξυπηρετητή τις διεργασίες όπου η χρησιμοποίηση του επεξεργαστή ξεπέρασε ένα κατώφλι(εδώ lim = 30) καθώς και το ποσοστό που η διεργασία είχε ξεπεράσει αυτό το κατώφλι στη διάρκεια μιας ημέρας.

```
    val conf = new
SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_5",
conf)
    val rdd= sc.cassandraTable("spark", "z_proccesses")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24
    val lim : Float = 30

    val rdd2=
rdd.where("cpu>?",0.1).select("hostid","user","cpu","command").where(
"timestamp>=? and timestamp<=?",myfilter,current_time)

    val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("com
mand")),(if(x.getFloat("cpu")>=lim) 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect
```

```
    def uuid = java.util.UUID.randomUUID

    var myList =
List[(java.util.UUID,Int,Int,String,String,Float,Float)]()
    for (i<- z){
      if (i._2._1!=0){
        myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,lim,i._2._1.toFloat/
i._2._2) :: myList
      }
    }
    val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats5",SomeColumns
("uid"," timestamp","hostid","user","command","lim","value"))
```

- **Scala Script 6,7**

Για τα Scala Script 6,7 ισχύει ό,τι για τα 4,5 αντίστοιχα με τη διαφορά ότι αντί για τον επεξεργαστή τα δεδομένα αφορούν τη μνήμη. Αυτό γίνεται αντικαθιστώντας τη λέξη “cpu” με τη λέξη “ram”.

- **Scala Script 8**

```
    val conf = new
SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_8",
conf)
    val rdd= sc.cassandraTable("spark", "logs_1")

    val current_time=System.currentTimeMillis/1000.toInt
    val format = new java.text.SimpleDateFormat("yy-MM-dd")
    val mydate =format.format(new java.util.Date())
    val rdd2=
rdd.select("id","priority","service","host").where("date=?",mydate)

    val z  =
rdd2.map(x=>((x.getString("service"),x.getStringOption("host")),(if(x
.getString("priority")=="WARNING") 1 else
0,if(x.getString("priority")=="ERROR") 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2,x._3 +
y._3)).collect

    def uuid = java.util.UUID.randomUUID

    var myList =
List[(java.util.UUID,Int,String,Option[String],Int,Int,Int,Float,Floa
t)]()
    for (i<- z){
        myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._2._1,i._2._2,i._2._3,i._2
._1.toFloat/i._2._3,i._2._2.toFloat/i._2._3) :: myList
    }
    val x =
sc.parallelize(myList).saveToCassandra("spark","log_stats1",SomeColum
ns("uid","timestamp","service","host","warning","error","sum","warper
","errper"))
```

Το Scala Script 8 περνάει στην Cassandra σε καθημερινή βάση τον αριθμό και το ποσοστό των προειδοποιήσεων “warnings” και σφαλμάτων “errors” σε σχέση με τον αριθμό των πληροφοριών που παράγει το Openstack.

Όπως αναφέρθηκε και παραπάνω οι ρουτίνες φιλτράρουν τα δεδομένα για ανάλυση συγκεκριμένου χρονικού διαστήματος. Στην πλατφόρμα που δημιουργήσαμε οι παραπάνω ρουτίνες κάνουν αναλύσεις σε καθημερινή βάση με τη βοήθεια του cron.

```
0 0 * * * ~/./spark.sh
```

Σχήμα 36. Προγραμματισμένη εκτέλεση των προγραμμάτων της υπηρεσίας ανάλυσης δεδομένων

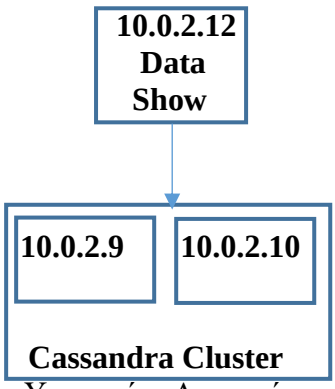
Η υπηρεσία Spark Cassandra Analysis όπως φαίνεται στην παραπάνω εικόνα κάθε μέρα στις 12 το βράδυ εκτελεί τα παραπάνω scala scripts.

Το “spark.sh” βρίσκεται στο παράρτημα Β.

Επίσης η κάθε ρουτίνα αποθηκεύει τα αποτελέσματα επεξεργασίας σε δικό του table.

6.1.5 Data Show

Η μονάδα Data Show είναι υπεύθυνη για την εμφάνιση των αποτελεσμάτων της ανάλυσης των δεδομένων που γίνεται, ώστε ο διαχειριστής της υποδομής να έχει μια εικόνα για την υποδομή.



Σχήμα 37. Σύμπλεγμα Υπηρεσίας Διεπαφής με τον Διαχειριστή

Συγκεκριμένα αποτελείται από 12 ρουτίνες οι οποίες παρουσιάζονται παρακάτω όσον αφορά τη χρήση τους.

results1_1.py

Η ρουτίνα *results1_1.py* εμφανίζει στο διαχειριστή της πλατφόρμας την ημερήσια χρήση της cpu για όλες τις εικονικές μηχανές. Το χρονικό βάθος εμφάνισης αποτελεσμάτων αλλάζει ανάλογα με τις ανάγκες του διαχειριστή. Κατά αυτόν τον τρόπο ο διαχειριστής μπορεί να αναζητήσει τα αποτελέσματα στο βάθος χρόνου που κρίνει αναγκαίο για την λήψη της απόφασης που καλείται να πάρει. Αυτό επιτυγχάνεται με τη χρήση μεταβλητής τιμής όσον αφορά το χρονικό διάστημα επιλογής των αποτελεσμάτων. Τα αποτελέσματα εμφανίζονται ομαδοποιημένα κατά εξυπηρετητή

και έπειτα κατά ημερομηνία ώστε να διευκολύνονται οι συγκρίσεις μεταξύ ημερομηνιών.

```
cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,key,timestamp  FROM
spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
      if (row.timestamp>= week_time and
row.key=='system.cpu.util[,user]'):

      a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).s
trftime('%d %b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('CPU AVG')
for x in a:
      print(x[1],x[2],x[3] ,'%')
session.cluster.shutdown()
```

Με παρόμοια λογική κινούνται και όλες οι υπόλοιπες ρουτίνες οι οποίες παρουσιάζονται στο παράρτημα Β.

results1_2.py
Εμφάνιση ημερήσιου μέσου όρου του δικτυακού όγκου που εισέρχεται απ' τις εικονικές μηχανές.

results1_3.py
Εμφάνιση ημερήσιου μέσου όρου του δικτυακού όγκου που εξέρχεται απ' τις εικονικές μηχανές.

results1_4.py
Εμφάνιση του μέσου όρου της διαθέσιμης μνήμης στη διάρκεια της ημέρας των εικονικών μηχανών.

results1_5.py
Εμφάνιση ημερήσιου μέσου όρου του δίσκου που χρησιμοποιείται των εικονικών μηχανών.

results2.py
Εμφάνιση του ποσοστού στη διάρκεια της ημέρας όπου η χρήση του επεξεργαστή στις εικονικές μηχανές ξεπερνάει το όριο 30%.

results3.py
Εμφάνιση των ενεργών σκανδαλών των εικονικών μηχανών στη διάρκεια της ημέρας. Επίσης εμφανίζεται το ποσοστό ενεργοποίησης στη διάρκεια της ημέρας.

results4.py
Το results4.py είναι υπεύθυνο για την εμφάνιση του ημερήσιου μέσου όρου χρήσης επεξεργαστή των εφαρμογών που τρέχουν στις εικονικές μηχανές.

results5.py
Το results5.py είναι υπεύθυνο για την εμφάνιση των εφαρμογών που η χρήση του επεξεργαστή ξεπερνάει το 30% καθώς και το ποσοστό στη διάρκεια ζωής τους που ξεπερνάνε αυτό το νούμερο.

results6.py
Το results6.py είναι υπεύθυνο για την εμφάνιση του ημερήσιου μέσου όρου χρήσης της μνήμης των εφαρμογών που τρέχουν στις εικονικές μηχανές.

results7.py
Το results5.py είναι υπεύθυνο για την εμφάνιση των εφαρμογών που η χρήση της μνήμης ξεπερνάει το 30% καθώς και το ποσοστό στη διάρκεια ζωής τους που ξεπερνάνε αυτό το νούμερο.

results8.py
Εμφάνιση του αριθμού των ειδοποιήσεων και των σφαλμάτων των υπηρεσιών που παράγει το Openstack.

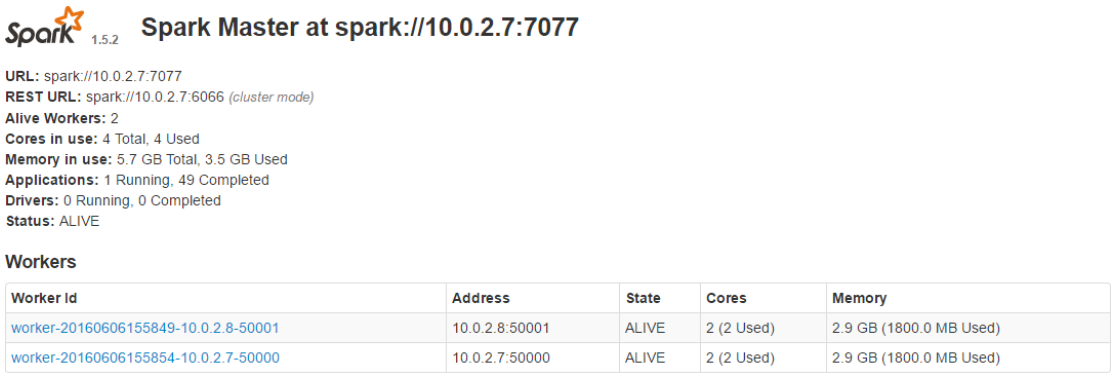
Όλες τις παραπάνω ρουτίνες τις τρέχει ο διαχειριστής της υποδομής όποτε το επιθυμεί με σκοπό την επίβλεψη του συνόλου της υποδομής.

6.1.6 Cassandra

Για να λειτουργήσουν όλα τα παραπάνω προγράμματα που δημιουργήθηκαν είναι απαραίτητη η δημιουργία των κατάλληλων tables, όπως και η δημιουργία του κατάλληλου keyspace.Το keyspace που δημιουργήθηκε έχει replication factor με τιμή 2, ώστε σε περίπτωση που ο ένας απ' τους δύο servers πάθει ζημιά να υπάρχει ένα αντίγραφο για να ανακτηθούν τα δεδομένα που χρειάζονται. Οι ρουτίνες δημιουργίας του keyspace και όλων των tables που δημιουργήθηκαν παρουσιάζονται στο παράρτημα Β.

6.2 Αποτελέσματα πειραματικής διαδικασίας

Στην παρούσα ενότητα εμφανίζονται τα αποτελέσματα της πειραματικής διαδικασίας. Αρχικά εμφανίζονται κάποια στοιχεία της υποδομής εν λειτουργία, μετέπειτα τα αποτελέσματα της ανάλυσης όπως εμφανίζονται στο χρήστη και τέλος τα στοιχεία αυτά σε γραφήματα.



Spark 1.5.2 **Spark Master at spark://10.0.2.7:7077**

URL: spark://10.0.2.7:7077
REST URL: spark://10.0.2.7:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 4 Total, 4 Used
Memory in use: 5.7 GB Total, 3.5 GB Used
Applications: 1 Running, 49 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20160606155849-10.0.2.8-50001	10.0.2.8:50001	ALIVE	2 (2 Used)	2.9 GB (1800.0 MB Used)
worker-20160606155854-10.0.2.7-50000	10.0.2.7:50000	ALIVE	2 (2 Used)	2.9 GB (1800.0 MB Used)

Σχήμα 38. Spark κατά τη διάρκεια εκτέλεσης

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160608002450-0064	analysis_8	4	1800.0 MB	2016/06/08 00:24:50	ubuntu	FINISHED	10 s
app-20160608002108-0063	analysis_7	4	1800.0 MB	2016/06/08 00:21:08	ubuntu	FINISHED	3.7 min
app-20160608001727-0062	analysis_6	4	1800.0 MB	2016/06/08 00:17:27	ubuntu	FINISHED	3.6 min
app-20160608001344-0061	analysis_5	4	1800.0 MB	2016/06/08 00:13:44	ubuntu	FINISHED	3.7 min
app-20160608001001-0060	analysis_4	4	1800.0 MB	2016/06/08 00:10:01	ubuntu	FINISHED	3.7 min
app-20160608000946-0059	analysis_3	4	1800.0 MB	2016/06/08 00:09:46	ubuntu	FINISHED	11 s
app-20160608000453-0058	analysis_2	4	1800.0 MB	2016/06/08 00:04:53	ubuntu	FINISHED	4.8 min
app-20160608000004-0057	analysis_1	4	1800.0 MB	2016/06/08 00:00:04	ubuntu	FINISHED	4.8 min

Σχήμα 39. Ημερήσια ανάλυση της μονάδας Spark Cassandra Analysis

```
ubuntu@cassandra1:~$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens       Owns    Host ID                               Rack
UN  10.0.2.9   12.65 GB   256          ?       bdc3b21e-3d1d-4231-bf4b-837cda6a9818  rack1
UN  10.0.2.10  12.34 GB   256          ?       5a99d84c-4a3a-4836-be36-2c85fc4d7743  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

Σχήμα 40. Στοιχεία του Cassandra Cluster

```
Table: z
SSTable count: 3
Space used (live): 5869287402
Table: z_proccesses
SSTable count: 7
Space used (live): 7708667347
Table: z_triggers2
SSTable count: 1
Space used (live): 2784201
Table: logs_1
SSTable count: 1
Space used (live): 1069059
```

Σχήμα 41. Μεγέθη βασικών πινάκων του Cassandra Cluster (σε bytes)

```
ubuntu@dataservice:~$ ps ux|grep ./read_log
ubuntu    8750  2.5  0.4 464952 20100 ?        Sl   Jun05 349:20 /usr/bin/python3.4 ./read_log5.py
ubuntu   11677  1.7  0.5 464892 22240 ?        Sl   Jun04 271:50 /usr/bin/python3.4 ./read_log2.py
ubuntu   11794  1.7  0.4 464944 20196 ?        Sl   Jun04 266:44 /usr/bin/python3.4 ./read_log3.py
ubuntu   11938  1.7  0.5 464888 22136 ?        Sl   Jun04 268:41 /usr/bin/python3.4 ./read_log4.py
ubuntu   12147  1.7  0.5 464952 22132 ?        Sl   Jun04 269:14 /usr/bin/python3.4 ./read_log6.py
ubuntu   12253  1.7  0.6 470120 28252 ?        Sl   Jun04 273:52 /usr/bin/python3.4 ./read_log7.py
ubuntu   12321  1.7  0.5 464888 22048 ?        Sl   Jun04 266:07 /usr/bin/python3.4 ./read_log8.py
ubuntu   12413  1.7  0.6 466552 24468 ?        Sl   Jun04 269:52 /usr/bin/python3.4 ./read_log9.py
ubuntu   23013  2.5  0.6 466820 26456 ?        Sl   Jun03 411:50 /usr/bin/python3.4 ./read_log1.py
```

Σχήμα 42. Log Cassandra Connector σε εκτέλεση

```
ubuntu@dataservice:~$ ps ux|grep ./read_zabbix.py
ubuntu   10885  0.0  0.0  10432   632 pts/32   S+   16:16   0:00 grep --color=auto ./read_zabbix.py
ubuntu   14999  3.7  0.6 469844 28040 pts/10   Sl   May28 932:00 /usr/bin/python3.4 ./read_zabbix.py
ubuntu@dataservice:~$ ps ux|grep ./zabbix_triggers.py
ubuntu   10929  0.0  0.0  10432   632 pts/32   S+   16:17   0:00 grep --color=auto ./zabbix_triggers.py
ubuntu   12067 10.0  0.6 468024 24488 ?        Rl   Jun01 1951:41 /usr/bin/python3.4 /home/ubuntu/./zabbix_triggers.py
ubuntu@dataservice:~$ ps ux|grep ./zabbix_script.py
ubuntu   10942  0.0  0.0  10432   632 pts/32   S+   16:17   0:00 grep --color=auto ./zabbix_script.py
ubuntu   18387  7.5  0.6 470492 27892 ?        Sl   Jun04 1087:23 /usr/bin/python3.4 /home/ubuntu/./zabbix_script.py
```

Σχήμα 43. Zabbix Cassandra Connector σε εκτέλεση

Όπως αναφέρθηκε και νωρίτερα ο διαχειριστής έχει τον έλεγχο του βάθους του ιστορικού εμφάνισης αποτελεσμάτων. Λόγω του όγκου των αποτελεσμάτων σε κάποιες από τις ρουτίνες αυτό το χρονικό διάστημα θα είναι διαφορετικό στις εικόνες που ακολουθούν.

```
ubuntu@dataservice:~$ ./results1_1.py 3
CPU AVG
04 Jun 16 10107 11.389668464660645 %
05 Jun 16 10107 15.25981616973877 %
06 Jun 16 10107 16.09544563293457 %
04 Jun 16 10113 4.6647562980651855 %
05 Jun 16 10113 3.5710127353668213 %
06 Jun 16 10113 10.278586387634277 %
04 Jun 16 10114 1.6922690868377686 %
05 Jun 16 10114 0.7968800067901611 %
06 Jun 16 10114 3.3233907222747803 %
04 Jun 16 10115 0.9610705971717834 %
05 Jun 16 10115 0.5186896920204163 %
06 Jun 16 10115 2.289595127105713 %
04 Jun 16 10116 8.555118560791016 %
05 Jun 16 10116 7.429680347442627 %
06 Jun 16 10116 13.619690895080566 %
04 Jun 16 10117 13.073136329650879 %
05 Jun 16 10117 13.036809921264648 %
06 Jun 16 10117 13.055130958557129 %
```

Σχήμα 44. Εμφάνιση αποτελεσμάτων – results1_1 για 3 μέρες

Στην παραπάνω εικόνα εμφανίζονται αποτελέσματα τριών ημερών όσων αφορά το μέσο όρο χρήσης του επεξεργαστή. Τα αποτελέσματα όπως είναι εμφανές περιλαμβάνουν όλους τους εξυπηρετητές που επιβλέπει η πλατφόρμα μας. Αλλάζοντας την τιμή 3 όπως φαίνεται παραπάνω μεταβάλλεται η χρονική διάρκεια.

Για τα ίδια αποτελέσματα με παραπάνω ,αλλά μόνο για τα στοιχεία της προηγούμενης ημέρας τρέχουμε την ίδια ρουτίνα αλλά αυτή τη φορά με παράμετρο 1 στην εκτέλεση.

```
ubuntu@dataservice:~$ ./results1_1.py 1
CPU AVG
06 Jun 16 10107 16.09544563293457 %
06 Jun 16 10113 10.278586387634277 %
06 Jun 16 10114 3.3233907222747803 %
06 Jun 16 10115 2.289595127105713 %
06 Jun 16 10116 13.619690895080566 %
06 Jun 16 10117 13.055130958557129 %
```
Σχήμα 45. Εμφάνιση αποτελεσμάτων – results1_1 για 1 μέρα

```
ubuntu@dataservice:~$ ./results1_2.py 2
NET IN AVG
05 Jun 16 10107 0.6562932729721069 Mbps
06 Jun 16 10107 0.6478733420372009 Mbps
05 Jun 16 10113 0.5537192821502686 Mbps
06 Jun 16 10113 2.4415531158447266 Mbps
05 Jun 16 10114 0.06827868521213531 Mbps
06 Jun 16 10114 2.036731243133545 Mbps
05 Jun 16 10115 0.2755546271800995 Mbps
06 Jun 16 10115 2.1102490425109863 Mbps
05 Jun 16 10116 0.5520468950271606 Mbps
06 Jun 16 10116 2.358999729156494 Mbps
05 Jun 16 10117 0.5179293155670166 Mbps
06 Jun 16 10117 0.5125313401222229 Mbps
```
Σχήμα 46. Εμφάνιση αποτελεσμάτων – results1_2

```
ubuntu@dataservice:~$ ./results1_3.py 2
NET OUT AVG
05 Jun 16 10107 0.7263304591178894 Mbps
06 Jun 16 10107 0.7155547738075256 Mbps
05 Jun 16 10113 0.5675655007362366 Mbps
06 Jun 16 10113 4.089000225067139 Mbps
05 Jun 16 10114 0.06780878454446793 Mbps
06 Jun 16 10114 0.14164072275161743 Mbps
05 Jun 16 10115 0.039960820227861404 Mbps
06 Jun 16 10115 0.08457042276859283 Mbps
05 Jun 16 10116 0.5778652429580688 Mbps
06 Jun 16 10116 4.167891979217529 Mbps
05 Jun 16 10117 0.5913682579994202 Mbps
06 Jun 16 10117 0.58594810962677 Mbps
```
Σχήμα 47. Εμφάνιση αποτελεσμάτων – results1_3

```
ubuntu@dataservice:~$ ./results1_4.py 2
MEM AVG AVAILABLE
05 Jun 16 10107 3.4084036350250244 GB
06 Jun 16 10107 3.3800017833709717 GB
05 Jun 16 10113 2.274883270263672 GB
06 Jun 16 10113 2.2485976219177246 GB
05 Jun 16 10114 2.828321695327759 GB
06 Jun 16 10114 2.6270668506622314 GB
05 Jun 16 10115 3.2307825088500977 GB
06 Jun 16 10115 3.1377100944519043 GB
05 Jun 16 10116 2.201622247695923 GB
06 Jun 16 10116 2.17960262298584 GB
05 Jun 16 10117 3.3706703186035156 GB
06 Jun 16 10117 3.370842933654785 GB
```

Σχήμα 48. Εμφάνιση αποτελεσμάτων – results1_4

```
ubuntu@dataservice:~$ ./results1_5.py 3
DISK AVG
04 Jun 16 10107 2.017038583755493 GB
05 Jun 16 10107 2.050236225128174 GB
06 Jun 16 10107 2.0428037643432617 GB
04 Jun 16 10113 12.115259170532227 GB
05 Jun 16 10113 13.163304328918457 GB
06 Jun 16 10113 12.639642715454102 GB
04 Jun 16 10114 20.748212814331055 GB
05 Jun 16 10114 21.77618980407715 GB
06 Jun 16 10114 21.88343048095703 GB
04 Jun 16 10115 19.958454132080078 GB
05 Jun 16 10115 21.081127166748047 GB
06 Jun 16 10115 21.539493560791016 GB
04 Jun 16 10116 12.180550575256348 GB
05 Jun 16 10116 13.246230125427246 GB
06 Jun 16 10116 12.754363059997559 GB
04 Jun 16 10117 11.16576099395752 GB
05 Jun 16 10117 11.977170944213867 GB
06 Jun 16 10117 12.631508827209473 GB
```

Σχήμα 49. Εμφάνιση αποτελεσμάτων – results1_5

```
ubuntu@dataservice:~$ ./results2.py 2
CPU UTILIZATION MORE THAN 30%
05 Jun 16 10107 0.0
06 Jun 16 10107 0.0
05 Jun 16 10113 0.0270700640976429
06 Jun 16 10113 0.18385545909404755
05 Jun 16 10114 0.0005790723371319473
06 Jun 16 10114 0.016271932050585747
05 Jun 16 10115 0.0
06 Jun 16 10115 0.0
05 Jun 16 10116 0.027678856626152992
06 Jun 16 10116 0.18860386312007904
05 Jun 16 10117 0.0
06 Jun 16 10117 0.0
```

Σχήμα 50. Εμφάνιση αποτελεσμάτων – results2

```
ubuntu@dataservice:~$ ./results3.py 7
enabled triggers
02 Jun 16 10107 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10107 Lack of free swap space on {HOST.NAME} 1.0
02 Jun 16 10113 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10113 Lack of free swap space on {HOST.NAME} 1.0
02 Jun 16 10114 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10114 Lack of free swap space on {HOST.NAME} 1.0
02 Jun 16 10115 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10115 Lack of free swap space on {HOST.NAME} 1.0
02 Jun 16 10116 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10116 Lack of free swap space on {HOST.NAME} 1.0
02 Jun 16 10117 Lack of free swap space on {HOST.NAME} 1.0
03 Jun 16 10117 Lack of free swap space on {HOST.NAME} 1.0
```

Σχήμα 51. Εμφάνιση αποτελεσμάτων – results3

```
ubuntu@dataservice:~$ ./results4.py 1
AVG CPU UTILIZATION PER PROCCESS
05 Jun 16 10114 java -Xms1024m -Xmx1024m -XX:R 161.5
06 Jun 16 10114 /usr/lib/jvm/java-8-oracle/jre 154.0
06 Jun 16 10114 /usr/lib/jvm/java-8-oracle/jre 209.0
06 Jun 16 10114 /usr/lib/jvm/java-8-oracle/jre 152.0
06 Jun 16 10114 /usr/lib/jvm/java-8-oracle/jre 185.0
06 Jun 16 10114 java -Xms1024m -Xmx1024m -XX:R 169.11538696289062
06 Jun 16 10115 /usr/lib/jvm/java-8-oracle/jre 160.0
06 Jun 16 10115 /usr/lib/jvm/java-8-oracle/jre 162.0
06 Jun 16 10115 /usr/lib/jvm/java-8-oracle/jre 151.0
06 Jun 16 10115 /usr/lib/jvm/java-8-oracle/jre 167.0
06 Jun 16 10115 /usr/lib/jvm/java-8-oracle/jre 179.0
```

Σχήμα 52. Εμφάνιση αποτελεσμάτων – results4

Στην παραπάνω εικόνα έχει γίνει μια έξτρα παραμετροποίηση, να εμφανίζει αποτελέσματα με χρήση της cpu παραπάνω από 150% λόγω του μεγάλου όγου των διεργασιών που εμφανίζονται.

```
ubuntu@dataservice:~$ ./results5.py 1
PERCENTAGE OF MORE THAN 30% CPU UTILIZATION PER DAY
06 Jun 16 10107 /usr/bin/python3 /us 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.1625615805387497
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.07999999821186066
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.1607142835855484
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.3384615480899811
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.08749999850988388
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.27272728085517883
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.28333333134651184
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.7142857313156128
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.09090909361839294
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.5303030014038086
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.1538461595773697
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.10606060922145844
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.2063492089509964
06 Jun 16 10114 /usr/lib/jvm/java-8- 0.27272728085517883
06 Jun 16 10114 /usr/lib/jvm/java-8- 1.0
```

Σχήμα 53. Εμφάνιση αποτελεσμάτων – results5

```
ubuntu@dataservice:~$ ./results6.py 1
AVG RAM USAGE PER PROCCESS
06 Jun 16 10113 /usr/lib/jvm/java-8-oracle/bin 76.08798217773438
06 Jun 16 10116 /usr/lib/jvm/java-8-oracle/bin 73.49331665039062
```

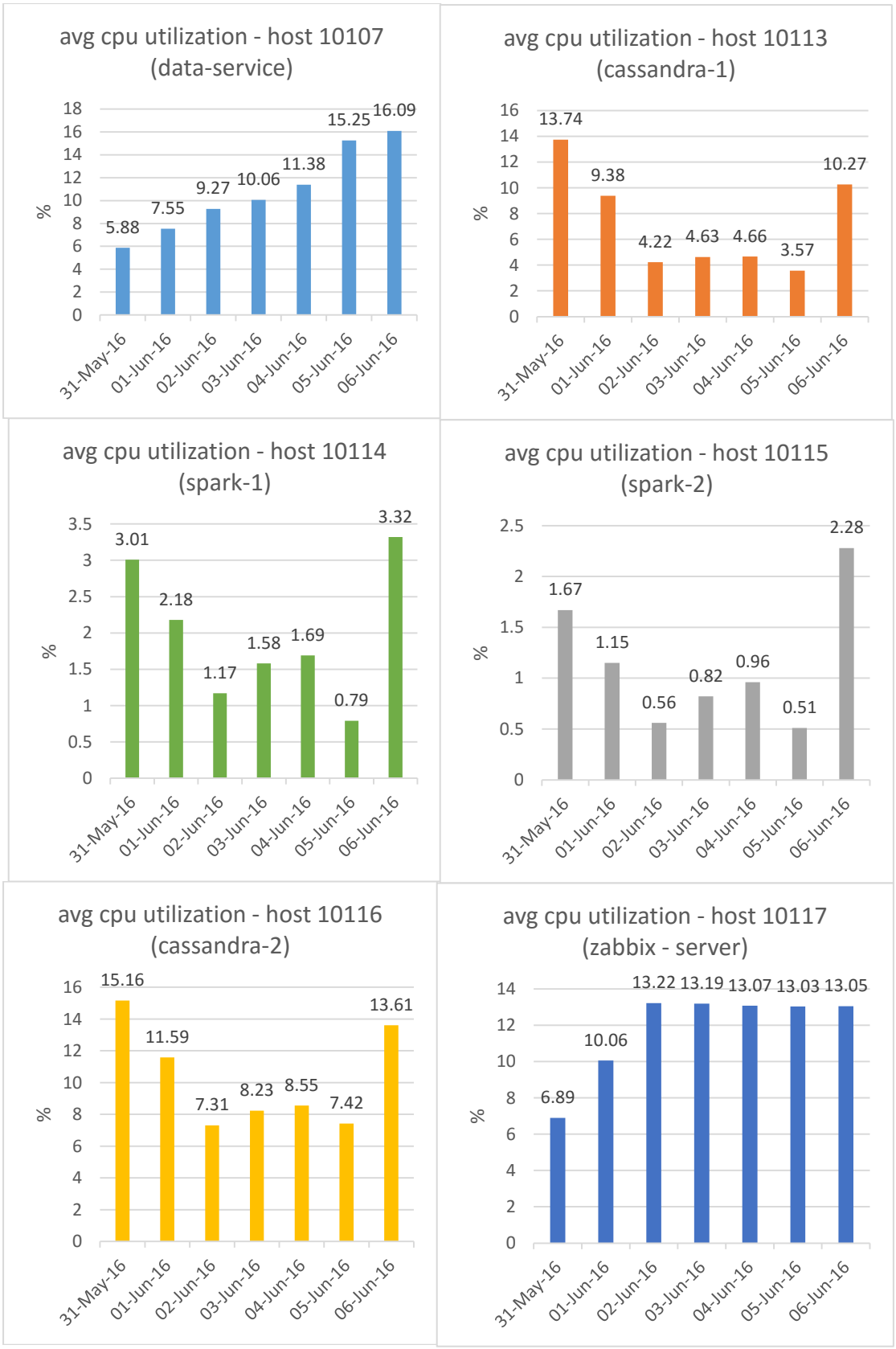
Σχήμα 54. Εμφάνιση αποτελεσμάτων – results6

```
ubuntu@dataservice:~$ ./results7.py 1
PERCENTAGE OF MORE THAN 30% RAM USAGE PER DAY
06 Jun 16 10113 /usr/lib/jvm/java-8-oracle/bin 1.0
06 Jun 16 10116 /usr/lib/jvm/java-8-oracle/bin 1.0
```

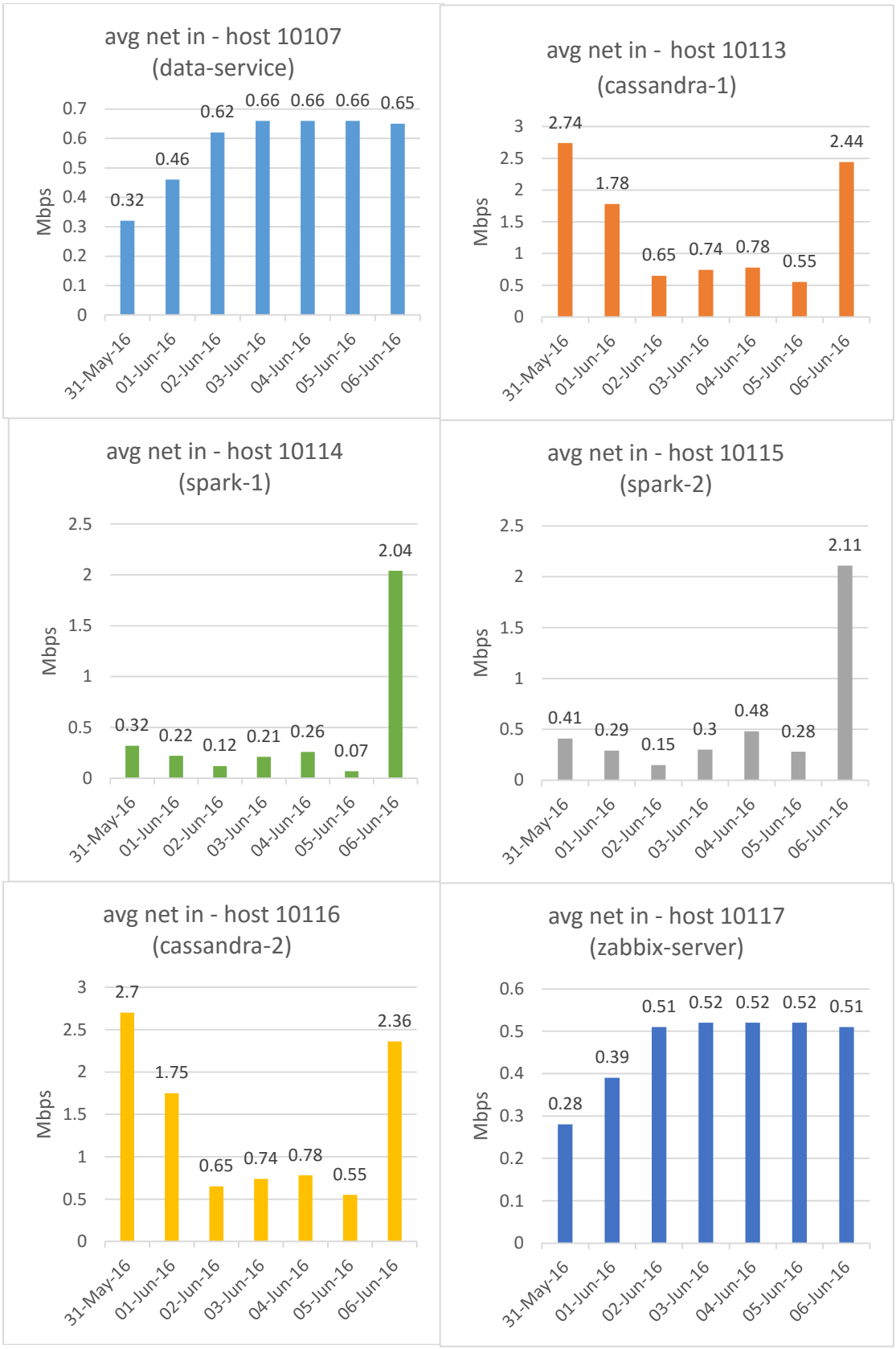
Σχήμα 55. Εμφάνιση αποτελεσμάτων – results7

```
ubuntu@dataservice:~$ ./results8.py 1
NUMBER OF ERRORS AND WARNINGS COMPARE THE INFO
14 Jun 16 192.168.2.12,10.0.0.10 nova.metadata.wsgi.server warnings: 0 errors: 0
14 Jun 16 192.168.2.11,10.0.0.10 nova.metadata.wsgi.server warnings: 0 errors: 0
14 Jun 16 192.168.2.5,10.0.0.10 nova.metadata.wsgi.server warnings: 0 errors: 0
14 Jun 16 None neutron.plugins.ml2.drivers.agent._common_agent warnings: 0 errors: 0
14 Jun 16 46647c1bba28423980cae16b3aa51979 oslo_config.cfg warnings: 1 errors: 0
14 Jun 16 None neutron.agent.linux.iptables_manager warnings: 6 errors: 0
14 Jun 16 9af721a51b9845cfb9af0939664c6e68 nova.api.openstack.compute.server_external_events warnings: 0 errors: 0
14 Jun 16 None neutron.agent.dhcp.agent warnings: 0 errors: 0
14 Jun 16 9af721a51b9845cfb9af0939664c6e68 nova.osapi_compute.wsgi.server warnings: 0 errors: 0
14 Jun 16 - nova.metadata.wsgi.server warnings: 0 errors: 0
14 Jun 16 None nova.consoleauth.manager warnings: 8794 errors: 0
14 Jun 16 46647c1bba28423980cae16b3aa51979 eventlet.wsgi.server warnings: 0 errors: 0
14 Jun 16 None nova.console.websocketproxy warnings: 0 errors: 0
14 Jun 16 None neutron.agent.securitygroups_rpc warnings: 0 errors: 0
14 Jun 16 46647c1bba28423980cae16b3aa51979 nova.osapi_compute.wsgi.server warnings: 0 errors: 0
```

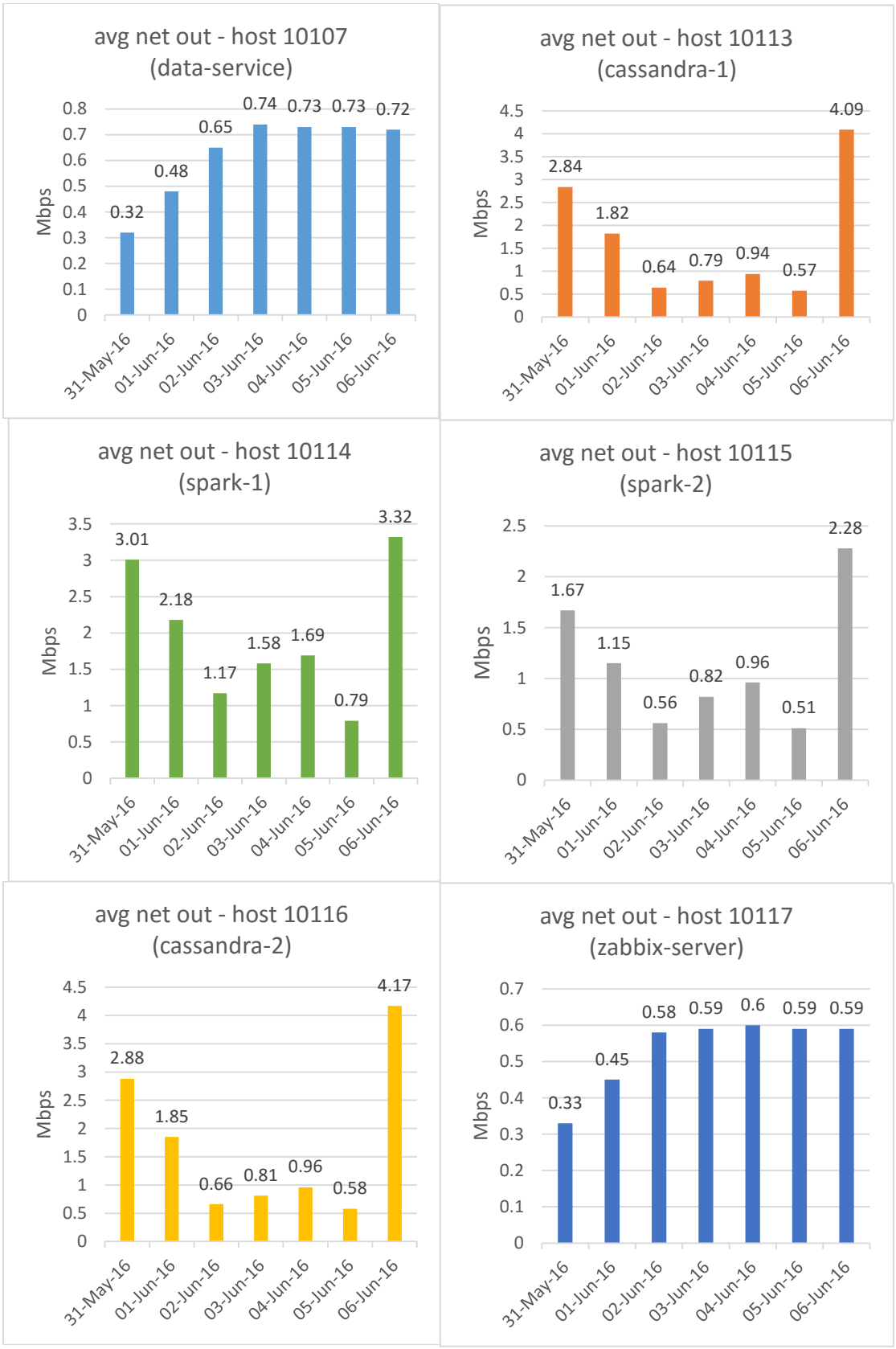
Σχήμα 56. Εμφάνιση αποτελεσμάτων – results8



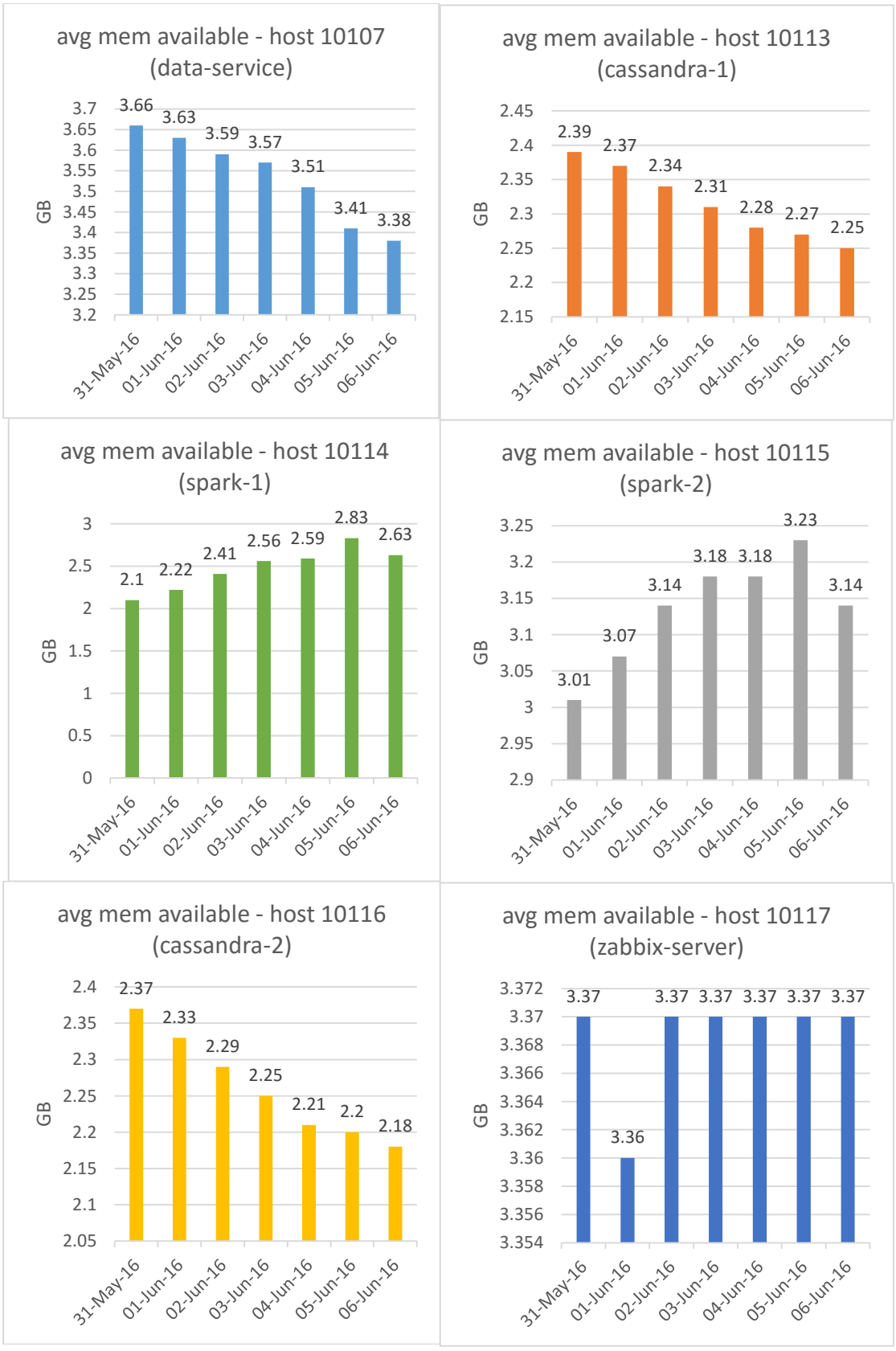
Σχήμα 57. Μέση χρήση επεξεργαστή σε μία εβδομάδα



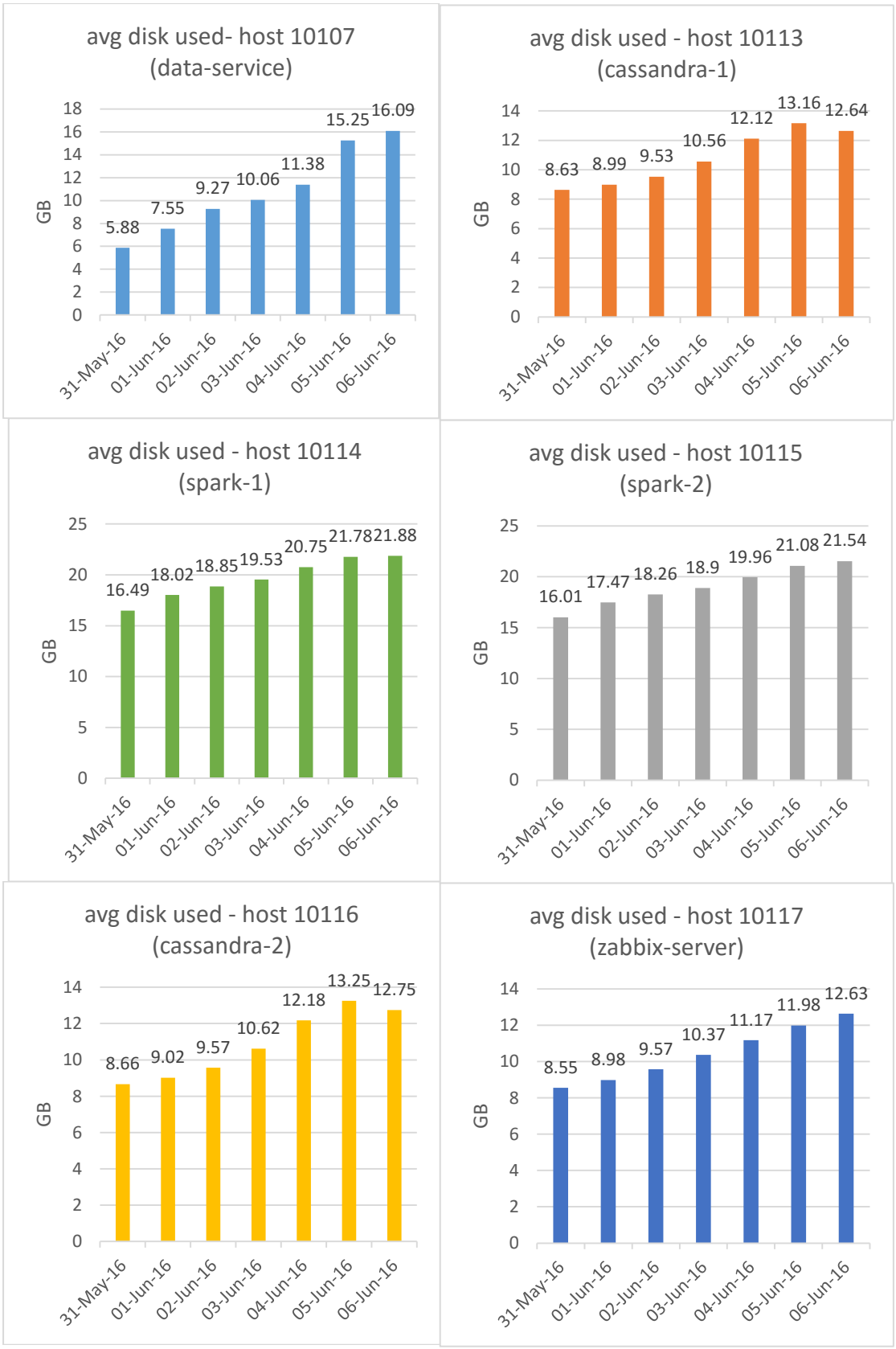
Σχήμα 58. Μέση κίνηση εισερχόμενη στο δίκτυο σε μία εβδομάδα



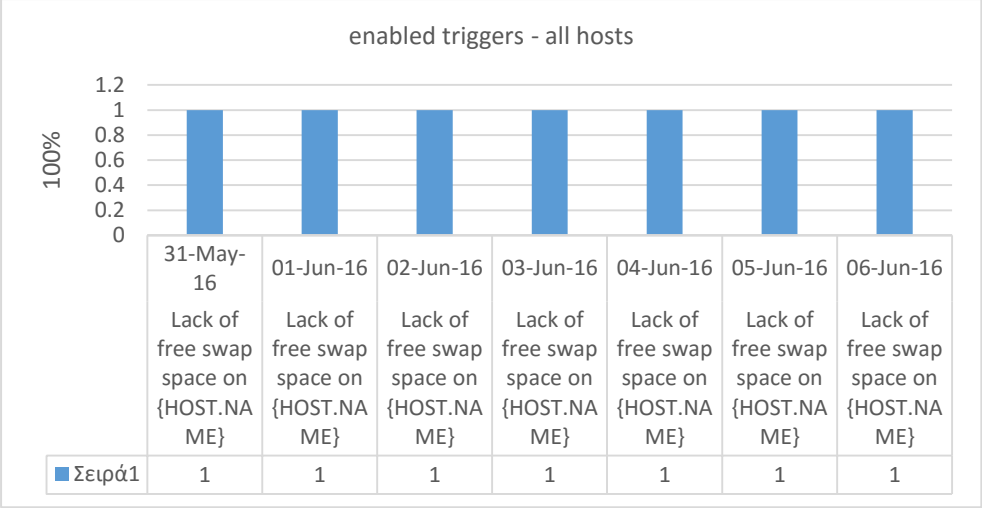
Σχήμα 59. Μέση κίνηση εξερχόμενη απ' το δίκτυο σε μία εβδομάδα



Σχήμα 60. Μέση διαθέσιμη μνήμη σε μία εβδομάδα

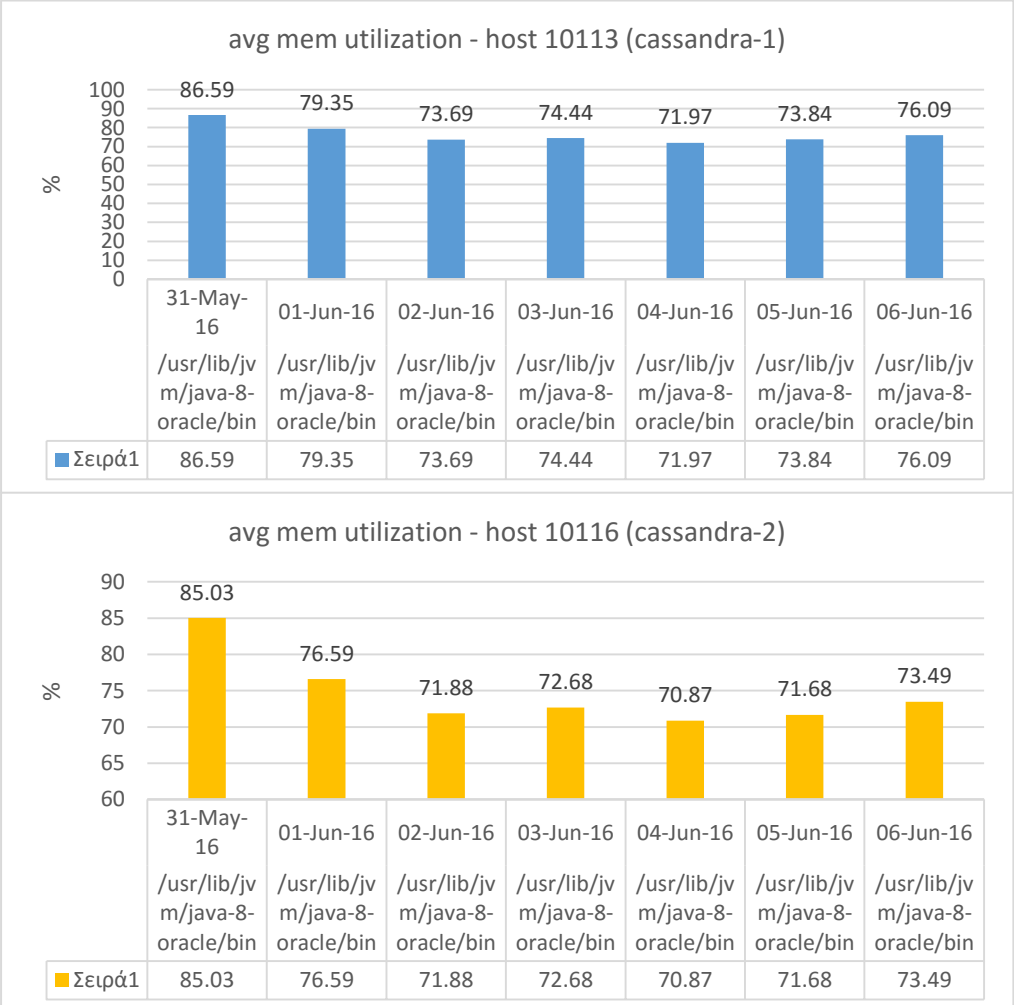


Σχήμα 61. Μέσος χρησιμοποιούμενος δίσκος σε μία εβδομάδα

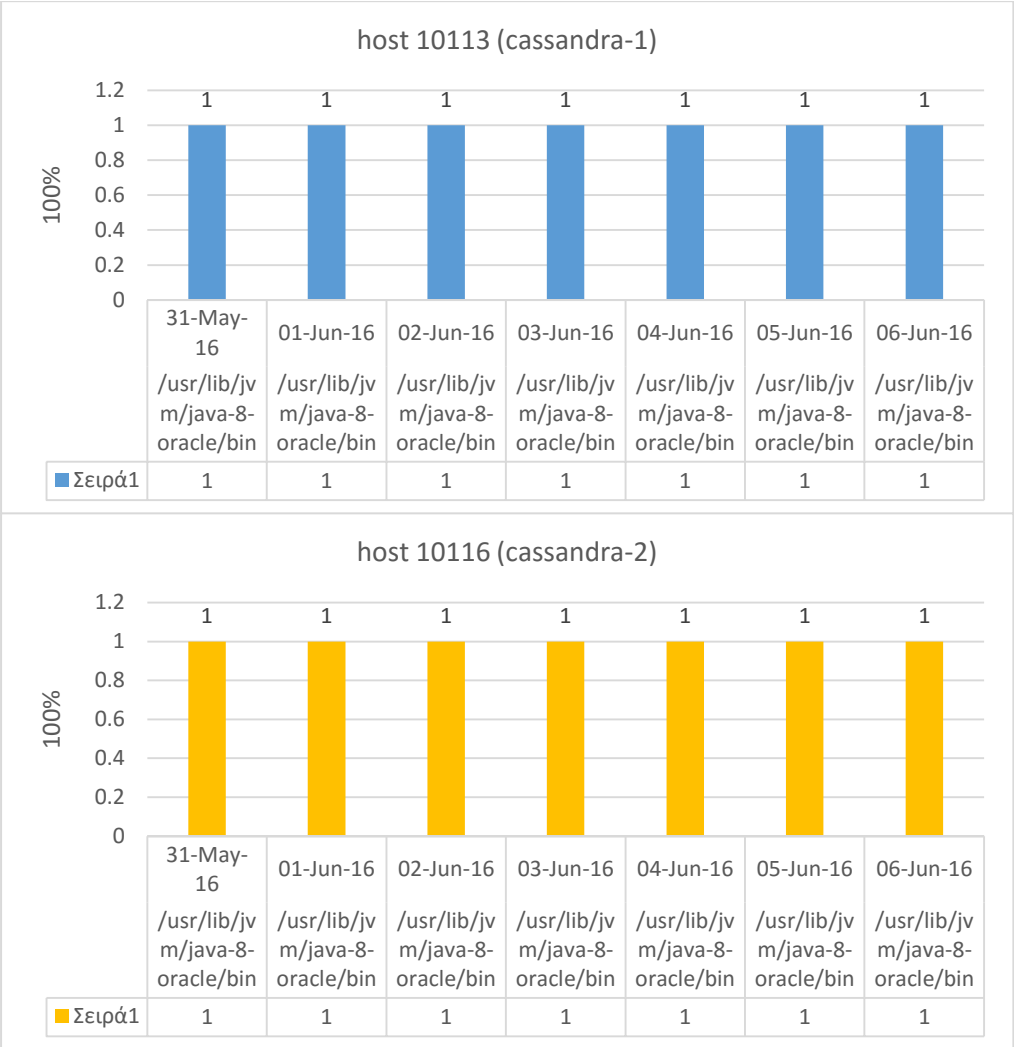


Σχήμα 62. Ενεργοποιημένες σκανδάλες και ποσοστό ενεργούς δράσης στη διάρκεια της ημέρας.

Η έλλειψη ελεύθερου χώρου ανταλαγής εμφανίζεται σε όλες τις εικονικές μηχανές και οφείλεται στο image των VMs.

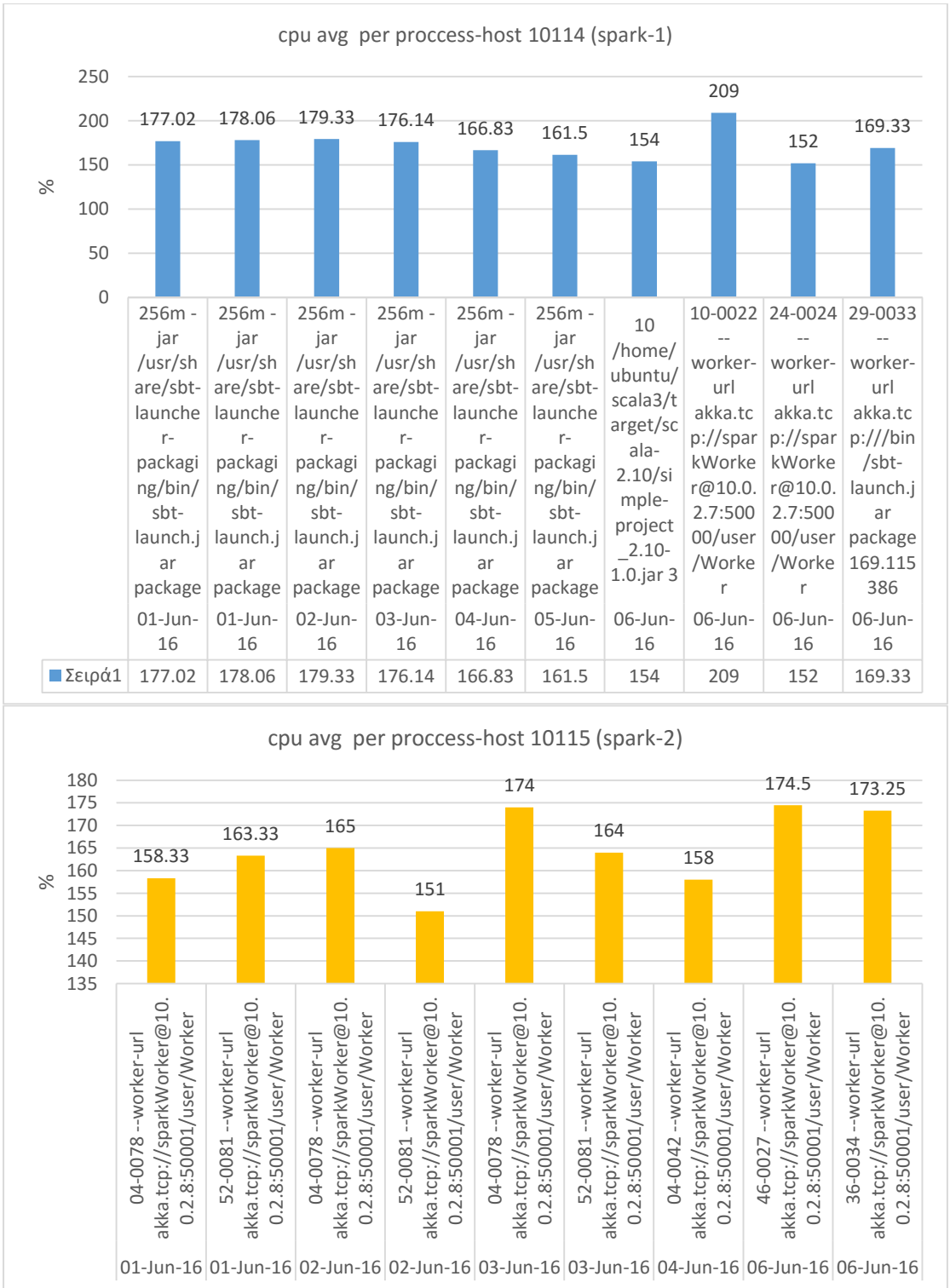


Σχήμα 63. Μέση χρησιμοποίηση μνήμης για εφαρμογές με χρήση περισσότερη από 50%



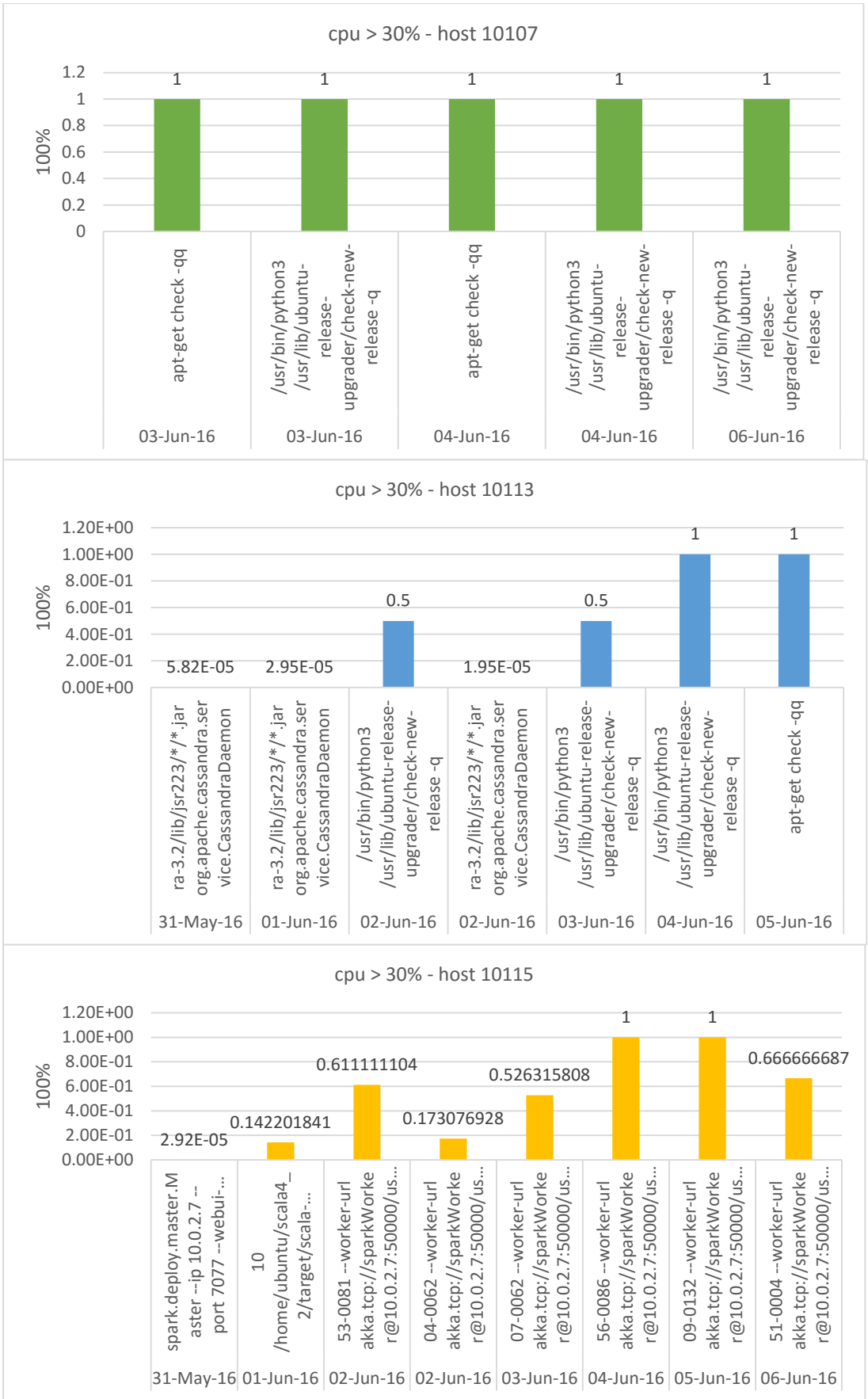
Σχήμα 64. Ποσοστό στη διάρκεια της μέρας για εφαρμογές με χρήση μνήμης περισσότερη από 30%

Όπως φαίνεται απ' τα δύο παραπάνω σχεδιαγράμματα οι μοναδικοί εξυπηρετητές που ξεπερνάνε το φράγμα του 30% είναι αυτοί που φιλοξενούν την κατανεμημένη βάση δεδομένων που χρησιμοποιούμε.



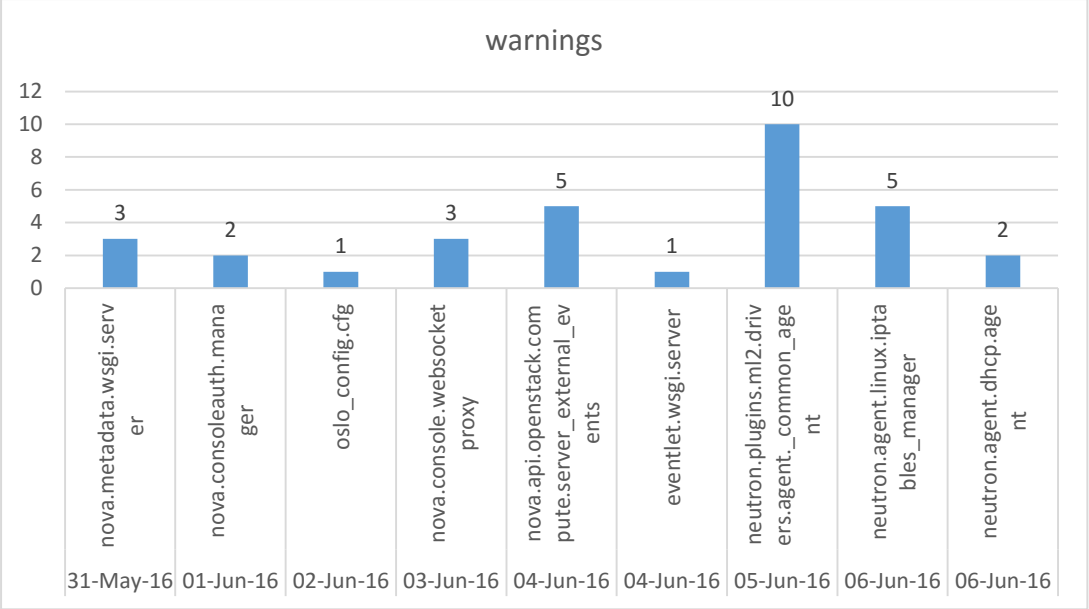
Σχήμα 65. Μέση χρησιμοποίηση επεξεργαστή από εφαρμογές κατά τη διάρκεια μιας εβδομάδας

Για λόγους χώρου εμφανίζονται οι εφαρμογές με χρήση της cpu στο 150%. Ο λόγος που το ποσοστό ξεπερνάει το 100% είναι ότι μέσω της εντολής ps aux για υπολογίζεται η συνολική χρήση για όλους τους πυρήνες. Επίσης απ΄τον εξυπηρετητή 10115 στο σχήμα 65 έχουν αφαιρεθεί αποτελέσματα για λόγους οπτικούς.

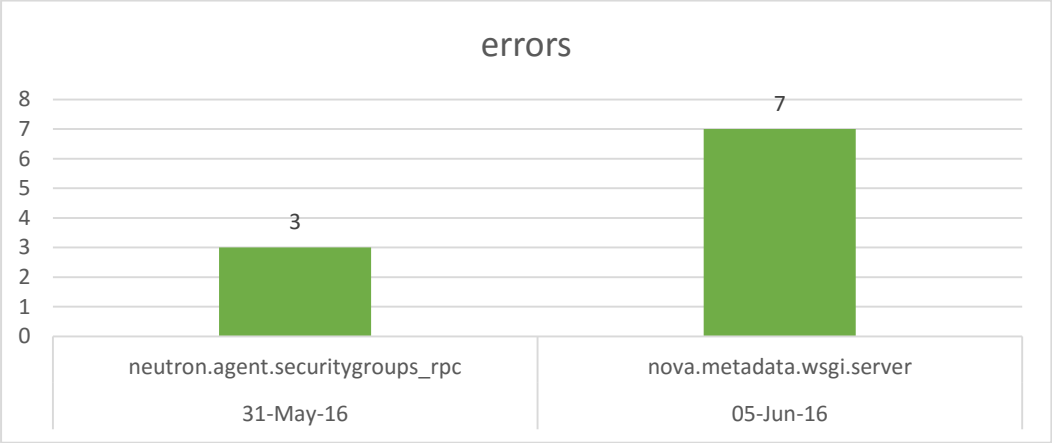


Σχήμα 66. Εφαρμογές οι οποίες χρησιμοποίησαν περισσότερο από 30% τον επεξεργαστή και το ποσοστό στη διάρκεια της ημέρας που είχαν χρήση πάνω από 30%

Σχήμα 67. Εφαρμογές οι οποίες χρησιμοποίησαν περισσότερο από 30% τον επεξεργαστή και το ποσοστό στη διάρκεια της ζωής τους που είχαν χρήση πάνω από 30%



Σχήμα 68. Αριθμός προειδοποιήσεων από υπηρεσίες του OpenStack



Σχήμα 69. Αριθμός σφαλμάτων από υπηρεσίες του OpenStack

Βιβλιογραφία

[1] Big Data: A Revolution That Will Transform How We Live, Work, and Think Paperback, by Viktor Mayer-Schönberger , Kenneth Cukier
[2] ICBNet Lab [Ιούνιος 2016]: http://www.icbnet.ntua.gr/
[3] Apache Hadoop [Ιούνιος 2016]: http://hadoop.apache.org/
[4] Apache Spark [Ιούνιος 2016]: http://spark.apache.org/
[5] Splunk [Ιούνιος 2016]: https://en.wikipedia.org/wiki/Splunk
[6] ELK stack [Ιούνιος 2016]: https://www.elastic.co/webinars/elk-stack-devops-environment
[7] IBM BigInsights [Ιούνιος 2016]: http://www-03.ibm.com/software/products/en/ibm-biginsights-for-apache-hadoop
[8]Microsof HDInsights [Ιούνιος 2016]: https://azure.microsoft.com/en-us/services/hdinsight/
[9] HDP [Ιούνιος 2016]: http://hortonworks.com/products/hdp/
[10] Cloudera CDH [Ιούνιος 2016]: https://www.cloudera.com/products/apache-hadoop/key-cdh-components.html
[11] 3D Data Managment [Ιούνιος 2016]: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf
[12] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008.
[13] NoSQL [Ιούνιος 2016]: https://en.wikipedia.org/wiki/NoSQL
[14] Cloud Computing [Ιούνιος 2016]:https://en.wikipedia.org/wiki/Cloud_computing
[15] OpenStack [Ιούνιος 2016]: https://en.wikipedia.org/wiki/OpenStack
[16] RestFul [Ιούνιος 2016]: https://en.wikipedia.org/wiki/Representational_state_transfer
[17] Zabbix [Ιούνιος 2016]: http://www.zabbix.com/
[18] Zabbix API [Ιούνιος 2016]: https://www.zabbix.com/documentation/3.0/manual/api
[19] JSON-RPC 2.0 [Ιούνιος 2016]: http://www.jsonrpc.org/specification
[20] Apache Cassandra [Ιούνιος 2016]: http://cassandra.apache.org/
[21] Apache Pig [Ιούνιος 2016]: https://pig.apache.org/
[22] Apache Hive [Ιούνιος 2016]: https://hive.apache.org/
[23] Java [Ιούνιος 2016]: https://www.java.com/en/
[24] Python [Ιούνιος 2016]: https://www.python.org/
[25] Amplab [Ιούνιος 2016]: https://amplab.cs.berkeley.edu/
[26] Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., J. Franklin M., Shenker S., & Stoica I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of NSDI 2012 at USENIX, pp. 15–28.
[27] Hadoop Yarn [Ιούνιος 2016]: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html
[28] Apache Mesos[Ιούνιος 2016]: http://mesos.apache.org/
[29] HDFS [Ιούνιος 2016]: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[30] MapR-FS[Ιούνιος 2016]: https://www.mapr.com/products/mapr-fs
[31] OpenStack Swift[Ιούνιος 2016]: https://wiki.openstack.org/wiki/Swift
[32] Amazon S3[Ιούνιος 2016]: https://aws.amazon.com/documentation/s3/

[33] Kudu [Ιούνιος 2016]: http://getkudu.io/
[34] Scala [Ιούνιος 2016]: http://www.scala-lang.org/
[35] R [Ιούνιος 2016]: https://en.wikipedia.org/wiki/R_(programming_language)
[36] Databricks [Ιούνιος 2016]: https://databricks.com/
[37] DataStax Python Driver for Apache Cassandra [Ιούνιος 2016]: https://github.com/datastax/python-driver
[38] Spark Scala Api[Ιούνιος 2016]: http://spark.apache.org/docs/latest/api/scala/index.html#package
[39] Spark Cassandra Connector[Ιούνιος 2016]: https://github.com/datastax/spark-cassandra-connector
[40] Requests [Ιούνιος 2016]: http://docs.python-requests.org/en/master/
[41] Cron[Ιούνιος 2016]: https://en.wikipedia.org/wiki/Cron
[42] Ren Li, Haibo Hu, Heng Li, Yunsong Wu, Jianxi Yang. MapReduce Parallel Programming Model: A State-of-the-Art Survey. In International Journal of Parallel Programming.2015

Βιβλιογραφία σχημάτων

1. Το τοπίο των Big Data το 2016 [Ιούνιος 2016]:
http://mattturck.com/2016/02/01/big-data-landscape/
2. Πλατφόρμα της IBM για ανάλυση Big Data [Ιούνιος 2016]:
http://www.slideshare.net/dianzi/value-proposition-for-big-data-isv-partners-0714
3. Πλατφόρμα της Hortonworks για ανάλυση Big Data [Ιούνιος 2016]:
http://hortonworks.com/products/hdp/
4. Πλατφόρμα της Cloudera για ανάλυση Big Data [Ιούνιος 2016]:
https://www.cloudera.com/products/apache-hadoop/apache-spark.html
5. Τομείς και χαρακτηριστικά των Big Data [Ιούνιος 2016]: http://olap.com/forget-big-data-lets-talk-about-all-data/
6. Μέτρηση λέξεων μέσω μοντέλου MapReduce [Ιούνιος 2016]:
https://cs.calvin.edu/courses/cs/374/exercises/12/lab/

7. Σύνοψη της αρχιτεκτονικής Cloud Computing [Ιούνιος 2016]:
http://www.gdv.com.au/cloud-computing.html

8. Στρώματα Cloud Computing αναπαριστώμενα σε μια στοίβα [Ιούνιος 2016]:
https://en.wikipedia.org/wiki/Cloud_computing#/media/File:Cloud_computing_layers.png
9. OpenStack αρχιτεκτονική κύριων συστατικών μονάδων [Ιούνιος 2016]:
https://www.openstack.org/software/
10. Apache Cassandra κλιμακωσιμότητα [Ιούνιος 2016]:
http://docs.datastax.com/en/cassandra/3.0/cassandra/cassandraAbout.html
11. Μια γραμμή σε μια οικογένεια στηλών [Ιούνιος 2016]:
http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/
12. Μια γραμμή σε μια υπερ-οικογένεια στηλών [Ιούνιος 2016]:
http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/
13. Spark εναντίων Hadoop [Ιούνιος 2016]: http://spark.apache.org/
14. Δομικές μονάδες Spark [Ιούνιος 2016]: http://spark.apache.org/
15. Spark αρχιτεκτονική [Ιούνιος 2016]: http://spark.apache.org/docs/latest/cluster-overview.html
16. Cassandra και Spark μέσω Connector [Ιούνιος 2016]:
http://www.slideshare.net/helenaedelson/streaming-bigdata-helenawebinarv3

Παράρτημα Α

Στο παράρτημα Α παρουσιάζονται οι παραμετροποιήσεις που έγιναν στα configuration files για την ανάπτυξη της πλατφόρμας.

Zabbix Agents σε όλους τους host

/etc/zabbix

zabbix_agentd.conf

EnableRemoteCommands=1
Server = 10.0.2.11

Spark Cluster

~/spark-1.5.2-bin-hadoop2.6/conf

10.0.2.7

spark-defaults.conf

```
spark.driver.port 50005
spark.blockManager.port 50006
spark.broadcast.port 50004
spark.replClassServer.port 50007
spark.fileserver.port 50008
spark.executor.port 50009
spark.driver.host 10.0.2.7
spark.executor.memory 1800m
spark.master                spark://10.0.2.7:7077
spark.eventLog.enabled true
spark.serializer            org.apache.spark.serializer.KryoSerializer
```

spark-env.sh

```
export SPARK_MASTER_IP=10.0.2.7
export SPARK_LOCAL_IP=10.0.2.7
export SPARK_WORKER_PORT=50000
```

10.0.2.8

```
export SPARK_MASTER_IP=10.0.2.8
export SPARK_LOCAL_IP=10.0.2.8
export SPARK_WORKER_PORT=50001
```

Cassandra Cluster

~/apache-cassandra-3.2/conf

cassandra.yaml

10.0.2.9

seeds: "10.0.2.9,10.0.2.10"
listen_address: 10.0.2.9
rpc_address: 10.0.2.9

10.0.2.10

seeds: "10.0.2.9,10.0.2.10"
listen_address: 10.0.2.10
rpc_address: 10.0.2.10

Παράρτημα Β

Στο παράρτημα Β εμφανίζονται όλοι οι κώδικες της πλατφόρμας στην πλήρη τους μορφή.

Zabbix Cassandra Connector scripts

Zabbix Script 1 – read_zabbix.py

```
#!/usr/bin/python3.4

from datetime import datetime
import time,json,requests,signal,sys,uuid
from cassandra.cluster import Cluster

class MyClient(object):
        def __init__(self):
                cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
                self.session = cluster.connect()

                self.url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
                self.headers = {"Content-Type":"application/json-rpc"}
                user = 'Admin'
                password = 'zabbix'
                self.login={
                   "jsonrpc": "2.0",
                   "method": "user.login",
                   "params": {
                           "user": user,
                           "password": password
                   },
                   "id": 1,
                   "auth": None
                }

                self.insert_statement  = self.session.prepare("""
                                        INSERT INTO spark.z (uid,timestamp,name,key,
value, itemid,hostid)
                                        VALUES (?,?,?,?,?,?,?);
                                        """)

        def authorization(self):

                r = requests.post(self.url,headers=self.headers,json=self.login)
                self.auth =(r.json())['result']

        def json_data(self):       ###json data for zabbix
                group="Thesis servers"
                self.memory = {            ###memory
                   "jsonrpc": "2.0",
```

```
                "method": "item.get",
                "params": {
                        "output": ["name","key_","lastvalue","lastclock","itemid","hostid"],
                        "group": group,
                        "application" : "Memory",
                },
                "auth": self.auth,
                "id": 1
            }

            self.cpu = {                ###cpu
                "jsonrpc": "2.0",
                "method": "item.get",
                "params": {
                        "output": ["name","key_","lastvalue","lastclock","itemid","hostid"],
                        "group": group,
                        "application" : "CPU",
                },
                "auth": self.auth,
                "id": 1
            }

            self.disk = {               ###disk
                        "jsonrpc": "2.0",
                        "method": "item.get",
                        "params": {
                                "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                                "group": group,
                                "application" : "Filesystems",
                        },
                        "auth": self.auth,
                        "id": 1
                    }

            self.net = {                ###network
                        "jsonrpc": "2.0",
                        "method": "item.get",
                        "params": {
                                "output":
["name","key_","lastvalue","lastclock","itemid","hostid"],
                                "group": group,
                                "application" : "Network interfaces",
                        },
                        "auth": self.auth,
                        "id": 1
                    }

        def get_data(self):

        #get cpu data
```

```
                r = requests.post(self.url,headers=self.headers,json=self.cpu)
                self.x=r.json()['result']

        #get ram data
                r = requests.post(self.url,headers=self.headers,json=self.memory)
                self.y=r.json()['result']

        #get disk data
                r = requests.post(self.url,headers=self.headers,json=self.disk)
                self.z=r.json()['result']

        #get network data
                r = requests.post(self.url,headers=self.headers,json=self.net)
                self.w=r.json()['result']

                result=self.x+self.y+self.z+self.w

                for w in result:

        self.session.execute(self.insert_statement,[uuid.uuid1(),int(w['lastclock']),w['name'],
w['key_'],w['lastvalue'],int(w['itemid']),int(w["hostid"])])

        def end_session(self):
                print('\nSession Closed')
                self.session.cluster.shutdown()
                sys.exit()

def main():
        try:
                client = MyClient()
                while True:
                        start_time = time.time()            ###start time of execution
                        client.authorization()
                        client.json_data()
                        client.get_data()
                        exec_time = time.time()-start_time        ###end time of execution
                        time.sleep(abs(5-exec_time))
        except KeyboardInterrupt:
                client.end_session()

if __name__ == "__main__":
        main()
```

Zabbix Script 2 – zabbix_triggers.py

```
#!/usr/bin/python3.4

from datetime import datetime
```

```
import time,json,requests,signal,sys,uuid
from cassandra.cluster import Cluster

class MyClient(object):
        def __init__(self):
                cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
                self.session = cluster.connect()

                self.url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
                self.headers = {"Content-Type":"application/json-rpc"}
                user = 'Admin'
                password = 'zabbix'
                self.login={
                   "jsonrpc": "2.0",
                   "method": "user.login",
                   "params": {
                           "user": user,
                           "password": password
                   },
                   "id": 1,
                   "auth": None
                }
                self.group="Thesis servers"

                self.insert_statement  = self.session.prepare("""
                                        INSERT INTO spark.z_triggers2
(uid,timestamp,triggerid,description,priority,lastchange,hostid,value)
                                        VALUES (?,?,?,?,?,?,?,?);
                                        """)

        def authorization(self):

                r = requests.post(self.url,headers=self.headers,json=self.login)
                self.auth =(r.json())['result']

        def json_data(self):        ###json data for zabbix
                self.trigger = {
                   "jsonrpc": "2.0",
                   "method": "trigger.get",
                   "params": {
                           "output": [
                   "triggerid",
                   "description",
                   "priority",
                             "lastchange",
                             "value",
                ],
                           "group": self.group,
                   },
                   "auth": self.auth,
                   "id": 1,
```

```
                }

        def get_data(self):

                r = requests.post(self.url,headers=self.headers,json=self.trigger)
                result=r.json()['result']

                for w in result:
                        host = {
                                "jsonrpc": "2.0",
                                "method": "host.get",
                                "params": {
                                        "output": "hostid",
                                        "triggerids" : w['triggerid'],
                                },
                                "auth": self.auth,
                                "id": 1,
                        }

                                r = requests.post(self.url,headers=self.headers,json=host)
        hostid=r.json()['result'][0]['hostid']

self.session.execute(self.insert_statement,[uuid.uuid1(),int(time.time()),int(w['triggerid']),w[
'description'],int(w['priority']),int(w['lastchange']),int(hostid),int(w[value])])

        def end_session(self):
                print('\nSession Closed')
                self.session.cluster.shutdown()
                sys.exit()


def main():
        try:
                client = MyClient()
                while True:
                        start_time = time.time()            ###start time of execution
                        client.authorization()
                        client.json_data()
                        client.get_data()
                        exec_time = time.time()-start_time        ###end time of execution
                        time.sleep(abs(5-exec_time))
        except KeyboardInterrupt:
                client.end_session()

if __name__ == "__main__":
        main()
```

Zabbix Script 3 - zabbix_script.py

```
#!/usr/bin/python3.4

from datetime import datetime
import time,json,requests,signal,sys,uuid
from cassandra.cluster import Cluster

class MyClient(object):
        def __init__(self):

                cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
                self.session = cluster.connect()

                self.url = "http://10.0.2.11/zabbix/api_jsonrpc.php"
                self.headers = {"Content-Type":"application/json-rpc"}
                user = 'Admin'
                password = 'zabbix'
                self.login={
                   "jsonrpc": "2.0",
                   "method": "user.login",
                   "params": {
                           "user": user,
                           "password": password
                   },
                   "id": 1,
                   "auth": None
                }

                self.insert_statement  = self.session.prepare("""
                        INSERT INTO spark.z_proccesses
(uid,timestamp,hostid,user,pid,cpu,ram,starts,lasts,command)
                        VALUES (?,?,?,?,?,?,?,?,?,?);
                        """)

        def authorization(self):

                r = requests.post(self.url,headers=self.headers,json=self.login)
                self.auth =(r.json())['result']

        def myscript_exec(self):
                groupid = 8
                host = {
                        "jsonrpc": "2.0",
                        "method": "host.get",
                        "params": {
                                "output": ["hostid","host"],
                                "groupids" : groupid,
                        },
                        "auth": self.auth,
                        "id": 1,
```

```
                }
                r = requests.post(self.url,headers=self.headers,json=host)
                hostids=r.json()['result']
                for i in hostids:
                        script_exec = {
                                "jsonrpc": "2.0",
                                "method": "script.execute",
                                "params": {
                                        "hostid" : i['hostid'],
                                        "scriptid" : 5,
                           },
                           "auth": self.auth,
                           "id": 1
                        }
                        r = requests.post(self.url,headers=self.headers,json=script_exec)
                        script=r.json()['result']['value']

                        #print (i['hostid'])
                        script2=str(script).split('\n')

                        for j in range(2,len(script2)):
                                x=script2[j].split()
                                c= x[10:]
                                command = ' '.join(c)

        self.session.execute(self.insert_statement,[uuid.uuid1(),int(time.time()),int(i['hostid'
]),x[0],int(x[1]),float(x[2]),float(x[3]),x[8],x[9],command])


        def end_session(self):
                print('\nSession Closed')
                self.session.cluster.shutdown()
                sys.exit()

def main():
        try:
                client = MyClient()
                while True:
                        start_time = time.time()
                        client.authorization()
                        client.myscript_exec()
                        exec_time = time.time()-start_time
                        time.sleep(abs(5-exec_time))
        except KeyboardInterrupt:
                client.end_session()

if __name__ == "__main__":
        main()
```

Log Cassandra Connector

read_log1.py

```
#!/usr/bin/python3.4

import time,linecache,sys,signal,uuid
from cassandra.cluster import Cluster

filename = 'Server_Logs/nova_controller/nova-api.log.1'
x=1       #old lines
y=1       #new lines
cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()
insert_statement  = session.prepare("""
          INSERT INTO spark.logs_1 (uid,date,time,id,priority,service,host,info)
          VALUES (?,?,?,?,?,?,?,?);
      """)

while True:
          try:

                    with open(filename, 'r') as f:
                              x=y
                              y=sum(1 for _ in f)+1
                    if y<x:
                              x=1

                    for i in range(x,y):
                              l=linecache.getline(filename,i).rstrip()
                              l2=l.split()
                              session.execute(insert_statement,[uuid.uuid1(),l2[0],
                                        l2[1],l2[2],l2[3],l2[4],l2[6]," ".join(l2[7:])])
                    linecache.clearcache()
                    time.sleep(5)

          except KeyboardInterrupt:
                    print('Session closed')
                    session.cluster.shutdown()
                    sys.exit()
```

Ισχύει το ίδιο για τα εξής αρχεία με μόνη αλλαγή το filename

read_log2.py – filename='Server_Logs/glance_controller/glance-api.log.1'
read_log3.py – filename='Server_Logs/glance_controller/glance-registry.log.1'

read_log4.py

```
#!/usr/bin/python3.4

import time,linecache,sys,signal,uuid
from cassandra.cluster import Cluster

filename = 'Server_Logs/neutron_compute1/neutron-linuxbridge-agent.log.1'
x=1       #old lines
y=1       #new lines
cluster = Cluster(['10.0.2.9','10.0.2.10',9042])
session = cluster.connect()
insert_statement  = session.prepare("""
          INSERT INTO spark.logs_1 (uid,date,time,id,priority,service,host,info)
          VALUES (?,?,?,?,?,?,?,?);
     """)

while True:
          try:

                    with open(filename, 'r') as f:
                              x=y
                              y=sum(1 for _ in f)+1
                    if y<x:
                              x=1

                    for i in range(x,y):
                              l=linecache.getline(filename,i).rstrip()
                              l2=l.split()

          session.execute(insert_statement,[uuid.uuid1(),l2[0],l2[1],l2[2],l2[3],l2[4],l2[6],"
".join(l2[7:])])
                    linecache.clearcache()
                    time.sleep(5)

          except KeyboardInterrupt:
                    print('Session closed')
                    session.cluster.shutdown()
                    sys.exit()
```

Ισχύει το ίδιο για τα εξής αρχεία με μόνη αλλαγή το filename

read_log5.py – filename='Server_Logs/neutron_controller/l3-agent.log'
read_log6.py – filename='Server_Logs/glance_controller/dhcp-agent.log.1'
read_log7.py – filename='Server_Logs/nova_controller/nova-consoleauth.log.1'
read_log8.py – filename='Server_Logs/glance_controller/nova-novncproxy.log.1'
read_log9.py – filename='Server_Logs/glance_controller/nova-scheduler.log.1'

Spark Cassandra Analysis scripts

spark.sh

```
#!/bin/bash

for i in {1..8}
do
~/spark-1.5.2-bin-hadoop2.6/bin/./spark-submit --packages datastax:spark-cassandra-
connector:1.5.0-s_2.10 ~/scala$i/target/scala-2.10/simple-project_2.10-1.0.jar
done
```

Scala Script 1

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

    val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_1", conf)
    val rdd= sc.cassandraTable("spark", "z")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24

    val rdd2= rdd.select("hostid","value","itemid","key").where("timestamp>=? and
timestamp<=?",myfilter,current_time).filter(x=>x.getString("key")=="vm.memory.size[availa
ble]" || x.getString("key")== "system.cpu.util[,user]" || x.getString("key")=="net.if.in[eth0]"
|| x.getString("key")=="net.if.out[eth0]" || x.getString("key")== "vfs.fs.size[/,used]")
    val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getInt("itemid"),x.getString("key")),(x.getFloat("value"),1)
)).reduceByKey((x,y)=>(x._1 + y._1, x._2 + y._2)).collect

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,Int,String,Float)]()
    for (i<- z){
      myList  = (uuid,current_time.toInt,i._1._1,i._1._3,i._2._1/i._2._2) :: myList
     }
     val x = sc.parallelize(myList).saveToCassandra("spark","z_stats1",SomeColumns("uid","
timestamp","hostid","key","value"))

  }
}
```

Scala Script 2

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

    val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_2", conf)
    val rdd= sc.cassandraTable("spark", "z")

    val current_time=System.currentTimeMillis/1000.toInt
    val n = 1
    val myfilter = current_time-n*3600*24
    val lim : Float = 30

    val rdd2= rdd.select("hostid","itemid","value"," key").where("timestamp>=? and
timestamp<=?",myfilter,current_time).filter(x=> x.getString("key")==
"system.cpu.util[,user]")

    val z  = rdd2.map(x=>(x.getInt("hostid"),(if(x.getFloat("value")>=lim) 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect
    z.foreach(println)

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,Int,Float,Float)]()
    for (i<- z){
      myList  = (uuid,current_time.toInt,i._1,lim,i._2._1.toFloat/i._2._2.toFloat) :: myList
    }
    val x = sc.parallelize(myList).saveToCassandra("spark","z_stats2",SomeColumns("uid","
timestamp","hostid","lim","value"))

  }
}
```

Scala Script 3

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {
```

```
   val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
   val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_3", conf)
   val rdd= sc.cassandraTable("spark", "z_triggers2")

   val current_time=System.currentTimeMillis/1000.toInt
   val n = 1
   val myfilter = current_time-n*3600*24

   val rdd2= rdd.where("timestamp>=? and
timestamp<=?",myfilter,current_time).select("triggerid","description","priority","lastchange
","hostid","value")

  val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("description"),x.getInt("priority"),x.getInt("lastc
hange"),x.getInt("triggerid")),(if(x.getFloat("value")==1) 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect

  def uuid = java.util.UUID.randomUUID

  var myList = List[(java.util.UUID,Int,Int,String,Int,Int,Int,Float)]()
  for (i<- z){
    myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,i._1._4,i._1._5,i._2._1.toFloat/i._2._2) ::
myList
  }
  val x = sc.parallelize(myList).saveToCassandra("spark","z_stats3",SomeColumns("uid","
timestamp","hostid","description","priority","lastchange","triggerid","value"))

 }
}
```

Scala Script 4

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
 def main(args: Array[String]) {

   val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
   val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_4", conf)
   val rdd= sc.cassandraTable("spark", "z_proccesses")

   val current_time=System.currentTimeMillis/1000.toInt
   val n = 1
```

```
   val myfilter = current_time-n*3600*24

   val rdd2=
rdd.where("cpu>?",0.1).select("hostid","user","cpu","command").where("timestamp>=?
and timestamp<=?",myfilter,current_time)

   val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("command")),(x.getFloat("cp
u"),1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect

   def uuid = java.util.UUID.randomUUID

   var myList = List[(java.util.UUID,Int,Int,String,String,Float)]()
   for (i<- z){
     myList  = (uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,i._2._1/i._2._2) :: myList
   }
   val x = sc.parallelize(myList).saveToCassandra("spark","z_stats4",SomeColumns("uid","
timestamp","hostid","user","command","value"))

 }
}
```

Scala Script 5

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

   val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
   val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_5", conf)
   val rdd= sc.cassandraTable("spark", "z_proccesses")

   val current_time=System.currentTimeMillis/1000.toInt
   val n = 1
   val myfilter = current_time-n*3600*24
   val lim : Float = 30

   val rdd2=
rdd.where("cpu>?",0.1).select("hostid","user","cpu","command").where("timestamp>=?
and timestamp<=?",myfilter,current_time)

   val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("command")),(if(x.getFloat("
cpu")>=lim) 1 else 0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect
```

```
   def uuid = java.util.UUID.randomUUID

   var myList = List[(java.util.UUID,Int,Int,String,String,Float,Float)]()
   for (i<- z){
    if (i._2._1!=0){
      myList  = (uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,lim,i._2._1.toFloat/i._2._2) ::
myList
    }
   }
   val x = sc.parallelize(myList).saveToCassandra("spark","z_stats5",SomeColumns("uid","
timestamp","hostid","user","command","lim","value"))

 }
}
```

Scala Script 6

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

   val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
   val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_6", conf)
   val rdd= sc.cassandraTable("spark", "z_proccesses")

   val current_time=System.currentTimeMillis/1000.toInt
   val n = 1
   val myfilter = current_time-n*3600*24

   val rdd2=
rdd.where("ram>?",0.1).select("hostid","user","ram","command").where("timestamp>=?
and timestamp<=?",myfilter,current_time)

   val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("command")),(x.getFloat("ra
m"),1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect

   def uuid = java.util.UUID.randomUUID

   var myList = List[(java.util.UUID,Int,Int,String,String,Float)]()
   for (i<- z){
    myList  = (uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,i._2._1/i._2._2) :: myList
   }
```

```
   val x = sc.parallelize(myList).saveToCassandra("spark","z_stats6",SomeColumns("uid","
timestamp","hostid","user","command","value"))

 }
}
```

Scala Script 7

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

   val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
   val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_7", conf)
   val rdd= sc.cassandraTable("spark", "z_proccesses")

   val current_time=System.currentTimeMillis/1000.toInt
   val n = 1
   val myfilter = current_time-n*3600*24
   val lim : Float = 30

   val rdd2=
rdd.where("ram>?",0.1).select("hostid","user","ram","command").where("timestamp>=?
and timestamp<=?",myfilter,current_time)

   val z  =
rdd2.map(x=>((x.getInt("hostid"),x.getString("user"),x.getString("command")),(if(x.getFloat("
ram")>=lim) 1 else 0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2)).collect

   def uuid = java.util.UUID.randomUUID

   var myList = List[(java.util.UUID,Int,Int,String,String,Float,Float)]()
   for (i<- z){
    if (i._2._1!=0){
      myList  = (uuid,current_time.toInt,i._1._1,i._1._2,i._1._3,lim,i._2._1.toFloat/i._2._2) ::
myList
     }
   }
   val x =
sc.parallelize(myList).saveToCassandra("spark","z_stats7",SomeColumns("uid","timestamp",
"hostid","user","command","lim","value"))

 }
}
```

Scala Script 8

```
import com.datastax.spark.connector._
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object SimpleApp {
  def main(args: Array[String]) {

    val conf = new SparkConf(true).set("spark.cassandra.connection.host",
"10.0.2.9,10.0.2.10")
    val sc = new SparkContext("spark://10.0.2.7:7077", "analysis_8", conf)
    val rdd= sc.cassandraTable("spark", "logs_1")

    val current_time=System.currentTimeMillis/1000.toInt
    val format = new java.text.SimpleDateFormat("yy-MM-dd")
    val mydate =format.format(new java.util.Date())
    val rdd2= rdd.select("id","priority","service","host").where("date=?",mydate)

    val z  =
rdd2.map(x=>((x.getString("service"),x.getStringOption("host")),(if(x.getString("priority")=="
WARNING") 1 else 0,if(x.getString("priority")=="ERROR") 1 else
0,1))).reduceByKey((x,y)=>(x._1 + y._1,x._2 + y._2,x._3 + y._3)).collect

    def uuid = java.util.UUID.randomUUID

    var myList = List[(java.util.UUID,Int,String,Option[String],Int,Int,Int,Float,Float)]()
    for (i<- z){
       myList  =
(uuid,current_time.toInt,i._1._1,i._1._2,i._2._1,i._2._2,i._2._3,i._2._1.toFloat/i._2._3,i._2._2.
toFloat/i._2._3) :: myList
    }
    val x =
sc.parallelize(myList).saveToCassandra("spark","log_stats1",SomeColumns("uid","timestamp
","service","host","warning","error","sum","warper","errper"))

 }
}
```

Data Show scripts

results1_1.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime


cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,key,timestamp  FROM spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.key=='system.cpu.util[,user]'):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('CPU AVG')
for x in a:
        print(x[1],x[2],x[3] ,'%')
session.cluster.shutdown()
```

results1_2.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,key,timestamp  FROM spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.key=='net.if.out[eth0]'):
                i=i+1
```

```
        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('NET OUT AVG')
for x in a:
        print(x[1],x[2],x[3] /(1024*1024),'Mbps')
session.cluster.shutdown()
```

results1_3.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,key,timestamp  FROM spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.key=='net.if.in[eth0]'):
                i=i+1

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('NET IN AVG')
for x in a:
        print(x[1],x[2],x[3] /(1024*1024),'Mbps')
session.cluster.shutdown()
```

results1_4.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()
```

```
results = session.execute("SELECT hostid,value,key,timestamp  FROM spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.key=='vm.memory.size[available]'):
                i=i+1

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('MEM AVG')
for x in a:
        print(x[1],x[2],x[3] /(1024*1024*1024),'GB')

session.cluster.shutdown()
```

results1_5.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,key,timestamp  FROM spark.z_stats1 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.key=='vfs.fs.size[/,used]'):
                i=i+1

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))
a.sort(key=lambda x:(x[2],x[0]))

print('DISK USED AVG')
for x in a:
        print(x[1],x[2],x[3] /(1024*1024*1024),'GB')
session.cluster.shutdown()
```

results2.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,lim,timestamp  FROM spark.z_stats2 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.value))

a.sort(key=lambda x:(x[2],x[0]))
print('CPU UTILIZATION MORE THAN 30%')
for x in a:
        print(x[1],x[2],x[3])
session.cluster.shutdown()
```

results3.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime


cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT value,hostid,description,timestamp  FROM spark.z_stats3
")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.value!=0):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.description,row.value))
```

```
a.sort(key=lambda x:(x[2],x[0]))
print('enabled triggers')
for x in a:
        print(x[1],x[2],x[3],x[4])

session.cluster.shutdown()
```

results4.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime


cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,command,timestamp  FROM spark.z_stats4 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time and row.value>=50):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.command,row.value))

a.sort(key=lambda x:(x[2],x[0],x[3]))
print('AVG CPU UTILIZATION PER PROCCESS')
for x in a:
        print(x[1],x[2],x[3][0:30],x[4])
session.cluster.shutdown()
```

results5.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()
```

```
results = session.execute("SELECT hostid,value,lim,timestamp,command  FROM
spark.z_stats5 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.command,row.value))

a.sort(key=lambda x:(x[2],x[0]))
print('PERCENTAGE OF MORE THAN 30% CPU UTILIZATION PER DAY')
for x in a:
        print(x[1],x[2],x[3][0:20],x[4])
session.cluster.shutdown()
```

results6.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,command,timestamp  FROM spark.z_stats6 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time):# and row.value>=30):
                i=i+1

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.command,row.value))

a.sort(key=lambda x:(x[2],x[0],x[3]))
print('AVG RAM USAGE PER PROCCESS')
for x in a:
        print(x[1],x[2],x[3][0:30],x[4])
session.cluster.shutdown()
```

results7.py

```
#!/usr/bin/python3.4
```

```
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime


cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

results = session.execute("SELECT hostid,value,lim,timestamp,command  FROM
spark.z_stats7 ")
current_time =int(time.time())
week_time= current_time-int(sys.argv[1])*3600*24
a=[]
for row in results:
        if (row.timestamp>= week_time):

        a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d
%b %y'),row.hostid,row.command,row.value))

a.sort(key=lambda x:(x[2],x[0]))
print('PERCENTAGE OF MORE THAN 30% RAM USAGE PER DAY')
for x in a:
        print(x[1],x[2],x[3][0:30],x[4])
session.cluster.shutdown()
```

results8.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
import time,sys
from datetime import datetime

try:
    cluster = Cluster(['10.0.2.9','10.0.2.10'])
    session = cluster.connect()

    results = session.execute("SELECT timestamp,service,host,warning,error  FROM
spark.log_stats1 ")
    current_time =int(time.time())
    week_time= current_time-int(sys.argv[1])*3600*24
    a=[]
    for row in results:
            if (row.timestamp>= week_time):

a.append((row.timestamp,datetime.fromtimestamp(row.timestamp).strftime('%d %b
%y'),row.host,row.service,row.warning,row.error))
```

```
        a.sort(key=lambda x:(x[3],x[0]))
        print('NUMBER OF ERRORS AND WARNINGS ')
        for x in a:
            print(x[1],x[2],x[3],'warnings:',x[4],'errors:',x[5])
        session.cluster.shutdown()

    except KeyboardInterrupt:
        session.cluster.shutdown()
        print('session closed')
        sys.exit()
```

Cassandra scripts

create_keyspace.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("CREATE KEYSPACE spark WITH replication= {'class':'SimpleStrategy',
'replication_factor':2}")

print('keyspace spark created.')
session.cluster.shutdown()
```

create_table_spark.z.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z (
                    uid uuid PRIMARY KEY,
                    timestamp int ,
                    key text,
                    name text,
                    value text,
                    itemid int,
                    hostid int
                    );
            """)

print('schema spark.z created.')
session.cluster.shutdown()
```

create_table_spark.z_triggers2.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
```

```
cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_triggers2 (
                    uid uuid PRIMARY KEY,
                    timestamp int ,
                    triggerid int,
                    description text,
                    priority int,
                    lastchange int,
                    hostid int,
                    value int
                    );
          """)

print('schema spark.z_triggers2 created.')
session.cluster.shutdown()
```

create_table_spark.z_proccesses,py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_proccesses (
                    uid uuid PRIMARY KEY,
                    timestamp int ,
                    hostid int,
                    user text,
                    pid int,
                    cpu float,
                    ram float,
                    starts text,
                    lasts text,
                    command text
                    );
          """)

print('schema spark.z_proccesses created.')
session.cluster.shutdown()
```

create_table_spark.logs_1.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-
```

```
from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.logs_1 (
                    uid uuid PRIMARY KEY,
                    date text ,
                    time text,
                    id text,
                    priority text,
                    service text,
                    host text,
                    info text
                    );
          """)

print('schema spark.logs_1 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats1.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats1 (
uid uuid PRIMARY KEY,
                    timestamp int,
                    hostid int,
                    key text,
                    value float
                    );
          """)

print('schema spark.z_stats1 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats2.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster
```

```
cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats2 (
uid uuid PRIMARY KEY,
                    timestamp int,
                    hostid int,
                    lim float,
                    value float
                    );
          """)

print('schema spark.z_stats2 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats3.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats3 (
uid uuid PRIMARY KEY,
                    timestamp int,
                    hostid int,
                    description text,
                    priority int,
                    lastchange int,
                    triggerid int,
                    value float
                    );
          """)

print('schema spark.z_stats3 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats4.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
```

```
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats4 (
uid uuid PRIMARY KEY,
                timestamp int,
                hostid int,
                user text,
                command text,
                value float
                );
        """)

print('schema spark.z_stats4 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats5.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats5 (
uid uuid PRIMARY KEY,
                timestamp int,
                hostid int,
                user text,
                command text,
                lim float,
                value float
                );
        """)

print('schema spark.z_stats5 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats6.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()
```

```
session.execute("""CREATE TABLE spark.z_stats6 (
uid uuid PRIMARY KEY,
                timestamp int,
                hostid int,
                user text,
                command text,
                value float
                );
        """)

print('schema spark.z_stats6 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats7.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.z_stats7 (
uid uuid PRIMARY KEY,
                timestamp int,
                hostid int,
                user text,
                command text,
                lim float,
                value float
                );
        """)

print('schema spark.z_stats7 created.')
session.cluster.shutdown()
```

create_table_spark.z_stats8.py

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from cassandra.cluster import Cluster

cluster = Cluster(['10.0.2.9','10.0.2.10'])
session = cluster.connect()

session.execute("""CREATE TABLE spark.log_stats1 (
uid uuid PRIMARY KEY,
```

```
                timestamp int,
                host text,
                warning int,
                error int,
                sum int,
                warper float,
                errper float
                );
        """)

print('schema spark.log_stats1 created.')
session.cluster.shutdown()
```