ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Εφαρμογή Υψηλής Διαθεσιμότητας Συστάδας Εξυπηρετητών με Οριζόντια Κατακερμάτιση.

Ιωάννης Ε. Ταρατίτας

Επιβλέπουσα **Ρουσσοπούλου Μέμα,** Αναπληρωτής Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2017

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Εφαρμογή Υψηλής Διαθεσιμότητας Συστάδας Εξυπηρετητών με Οριζόντια Κατακερμάτιση

Ιωάννης Ε. Ταρατίτας
Α.Μ.: Μ1327

ΕΠΙΒΛΕΠΟΥΣΑ **Ρουσσοπούλου Μέμα,** Αναπληρωτής Καθηγητής ΕΚΠΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2017

ΠΕΡΙΛΗΨΗ

Οι παραδοσιακές σχεσιακές βάσεις, παρά την ευρεία αποδοχή τους, παρουσιάζουν αδυναμίες όταν καλούνται να διαχειριστούν ιδιαίτερα μεγάλο όγκο δεδομένων. Τα τελευταία χρόνια, η ραγδαία αύξηση του όγκου των δεδομένων έχει οδηγήσει στην ανάπτυξη νέων συστημάτων βάσεων δεδομένων. Προκειμένου να γίνει εφικτή η διαχείρισή, τα νέα συστήματα βάσεων δεδομένων διαφοροποιούνται από τα παραδοσιακά, εφαρμόζοντας οριζόντια κατακερμάτιση μεταξύ πολλαπλών κόμβων, προκειμένου να εξασφαλίσουν τη δυνατότητα οριζόντιας κλιμάκωσης για παράλληλη αποθήκευση και επεξεργασία δεδομένων.

Σκοπός αυτής της διπλωματικής εργασίας είναι η σχεδίαση και η εφαρμογή μιας υψηλής διαθεσιμότητας αρχιτεκτονικής συστάδας εξυπηρετητών MariaDB (Server Cluster) , χρησιμοποιώντας τον μηχανισμό αποθήκευσης δεδομένων (storage engine) Spider , καθώς και η σύγκριση των επιδόσεών τους με μια παραδοσιακή κεντρικοποιημένη αρχιτεκτονική, χρησιμοποιώντας το μετροπρόγραμμα SysBench.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Συστήματα Κατανεμημένων Βάσεις Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Υψηλή Διαθεσιμότητα, Συστάδα Εξυπηρετητών, Οριζόντια Κατακερμάτιση , Μηχανισμός Αποθήκευσης Αράχνης, Αφέντης - Σκλάβος

ABSTRACT

Traditional relational databases, despite their wide adoption, begin to show weaknesses when having to deal with very large amounts of data. The rapid growth of data over the last few years, has led to the emergence of new database management systems. In order to manage those data , the new database management systems are differentiated from the traditional ones, by sharding the data through multiple nodes and achieving linear scalability. in order to store and manage the data in parallel.

The goal of this thesis is to design and implement a high availability architecture of MariaDB Server Cluster using Spider Storage Engine and compare their performance to a traditional centralized architecture using the SysBench benchmark.

SUBJECT AREA: Distributed Database Systems

KEYWORDS: High Availability, Server Cluster, Sharding, Spider Storage Engine, Master -Slave

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την καθηγήτριά μου κα. Ρουσσοπούλου Μέμα για την εμπιστοσύνη που μου έδειξε , την υπομονή, την καθοδήγηση και τις πολύτιμες συμβουλές και την συνολικής της συμβολή για την ολοκλήρωση της εργασίας μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ 13

1. ΕΙΣΑΓΩΓΗ 14

1.1 Η εποχή των Μεγάλων Δεδομένων (Big Data) 14

1.2 Ορισμοί εννοιών 15

1.3 Πλεονεκτήματα Κατακερματισμού 17

1.4 Προβλήματα Κατακερματισμού 17

1.5 Ιδιότητες ACID 18

1.6 Οι κανόνες του Date 21

1.7 Θεώρημα CAP 22

1.8 Σκοπός 23

2. ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΩΝ 24

2.1 Συστάδα εξυπηρετητών Γαλέρα : Εισαγωγή 24
2.1.1 Χαρακτηριστικά συστάδας εξυπηρετητών Γαλέρα 24
2.1.2 Λειτουργία Συστάδας εξυπηρετητών Γαλέρα 25
2.1.3 Σύνδρομο Διάσπασης Εγκεφάλου (Split Brain Syndrome) 26
2.1.4 Πλεονεκτήματα συστάδας εξυπηρετητών Γαλέρα 26

2.2 Υψηλής Διαθεσιμότητας Εξυπηρετητής (HAProxy): Εισαγωγή 26
2.2.1 Λειτουργία HAProxy 27
2.2.2 Έλεγχοι υγείας για MySQL. 28
2.2.3 Αναγνώριση μη διαθεσιμότητας του κόμβου. 30
2.2.4 Διαχωρισμός ανάγνωσης / εγγραφής με HAProxy. 32

2.3 Διαθεσιμότητα του HAProxy μέσω του Keepalived. 32
2.3.1 Αναφορά Στατιστικών. 34
2.3.2 Πλεονεκτήματα Keepalived- HAProxy. 36

2.4 Μηχανισμός Αποθήκευσης Δεδομένων Spider: Εισαγωγή 36
2.4.1 Λειτουργία spider. 38
2.4.2 Συνήθης διατάξεις κόμβων spider. 40
2.4.3 Σενάρια χρήσης πινάκων spider. 41
2.4.4 Τήρηση αντιγράφων δεδομένων και spider. 44
2.4.5 Πλεονεκτήματα χρήσης μηχανισμού αποθήκευσης δεδομένων spider. 45

2.5 Μετροπρόγραμμα SysBench 45

3. ΠΕΡΙΓΡΑΦΗ ΠΕΙΡΑΜΑΤΟΣ 47

3.1 Εισαγωγή – Συνοπτική παρουσίαση του πειράματος. 47
3.1.1 Υψηλής διαθεσιμότητας συστάδα εξυπηρετητών πολλαπλών αφεντών. 47
3.1.2 Διάταξη αντιγραφής δεδομένων αφέντη – σκλάβου. 48
3.1.3 Οριζόντια κατακερμάτιση βάσης δεδομένων μέσω της μηχανής αποθήκευσης spider. 49

3.2 Υλοποίηση του πειράματός μας. 50
3.2.1 Υλοποίηση της διάταξης υψηλής διαθεσιμότητας 50
3.2.2 Υλοποίηση της διάταξης αφέντη - σκλάβου 57

3.2.3 Υλοποίηση της διάταξης κόμβων spider. 59

4. ΠΕΙΡΑΜΑΤΙΚΕΣ ΜΕΤΡΗΣΕΙΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ 66

4.1 Εισαγωγή – Συνοπτική παρουσίαση των μετρήσεων 66

4.2 Σκοπός 66
4.2.1 Προαπαιτούμενα 67
4.2.2 SysBench 67

4.3 Ερωτήματα που θα χρησιμοποιηθούν προς την βάση μας. 68
4.3.1 Ερώτημα Q1. 68
4.3.2 Ερώτημα Q2. 70
4.3.3 Ερώτημα Q3. 71
4.3.4 Ερώτημα Q4. 72

4.4 Συμπεράσματα 74

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ 75

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ 77

ΑΝΑΦΟΡΕΣ 78

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Σχήμα 1: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q1…….....69

Σχήμα 2: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q2…….....70

Σχήμα 3: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q3…….....72

Σχήμα 4: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q4……...73

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Εικόνα 1: Εταιρείες που χρησιμοποιούν κατακερματισμό των δεδομένων [5]. 14
Εικόνα 2: Παράδειγμα οριζόντιου κατακερματισμού 16
Εικόνα 3: Παράδειγμα κατακόρυφου κατακερματισμού 16
Εικόνα 4: Παράδειγμα διάταξης πολλαπλών αφεντών και σκλάβων. 17
Εικόνα 5: ACID ιδιότητες [10].....20
Εικόνα 6:Το θεώρημα CAP [13].....23
Εικόνα 7: Βασική αρχιτεκτονική συστάδας εξυπηρετητών MariaDB. 24
Εικόνα 8: Διάταξη εξισορρόπησης φορτίου. 27
Εικόνα 9: Παράδειγμα Αρχιτεκτονικής HAProxy 28
Εικόνα 10: Αρχείο Διαμόρφωσης /etc/xinetd.d/mysqlchk.....29
Εικόνα 11:Τμήμα του αρχείου κώδικα /usr/bin/clustercheck 30
Εικόνα 12: Τμήμα του αρχείου /etc/haproxy/haproxy.cfg. 30
Εικόνα 13: Αρχείο Διαμόρφωσης /etc/keepalived/keepalived.conf 33
Εικόνα 14: Προσθήκη keepalived στην αρχιτεκτονική μας.....34
Εικόνα 15: Σελίδα στατιστικών HAProxy2.....35
Εικόνα 16: Ανάθεση VIP στον HAProxy2 35
Εικόνα 17: Αρχιτεκτονική μηχανισμών MariaDB 37
Εικόνα 18: Εγκατάσταση μηχανισμού αποθήκευσης δεδομένων Spider. 38
Εικόνα 19: Επιτυχής Εγκατάσταση μηχανισμού αποθήκευσης δεδομένων Spider. 38
Εικόνα 20: Λειτουργία μηχανισμού αποθήκευσης δεδομένων Spider.....39
Εικόνα 21: Μοντέλο νημάτων spider.....39
Εικόνα 22: Διατάξεις κόμβων spider: A- Federation και B-Federation HA 40
Εικόνα 23: Διατάξεις κόμβων spider: C-Sharding και D-Sharding HA 40
Εικόνα 24: Πίνακας στον backend1 και backend2 41
Εικόνα 25: Πίνακας συνδέσεων server στον spider 41
Εικόνα 26: Πίνακας spider περίπτωση χρήσης 1.....42
Εικόνα 27: Πίνακας spider περίπτωση χρήσης 2.....42
Εικόνα 28: Πίνακας spider περίπτωση χρήσης 3.....43
Εικόνα 29: Πίνακας spider περίπτωση χρήσης 4.....43
Εικόνα 30 : Διάταξη τήρησης αντιγράφων δεδομένων 1.....44
Εικόνα 31 : Διάταξη τήρησης αντιγράφων 2 44
Εικόνα 32 : Παράδειγμα δοκιμασίας δημιουργίας 3.500.000 εγγραφών μέσω του SysBench. 45
Εικόνα 33: Πίνακας sbtest 46
Εικόνα 34: Υλοποίηση πίνακα sbtest σε spider. 46
Εικόνα 35 : Υψηλής διαθεσιμότητας συστάδα εξυπηρετητών. 47

Εικόνα 36 : Διάταξη αφέντη - σκλάβου 49
Εικόνα 37: Συνολική διάταξη της οριζόντια κατακερματισμένης βάσης δεδομένων μας 50
Εικόνα 38. Πείραμα με HAProxy - VIP 51
Εικόνα 39:VirtualBox Host-Only Adapter 51
Εικόνα 40 : Ρυθμίσεις Δικτύου VM 51
Εικόνα 41. HA keepalived cluster 52
Εικόνα 42: Ρυθμίσεις στο αρχείο /etc/my.cnf.d/server.cnf 52
Εικόνα 43 : Επιτυχές αποτέλεσμα δημιουργίας συστάδας με δύο εξυπηρετητές 53
Εικόνα 44: Καρτέλα στατιστικών HAProxy 54
Εικόνα 45: Ο haproxy1 αναλαμβάνει την VIP 54
Εικόνα 46: Ο haproxy2 αναλαμβάνει την VIP 55
Εικόνα 47: Clustercheck mariadbcluster1 55
Εικόνα 48:Clustercheck mariadbcluster2 55
Εικόνα 49 :Telnet haproxy1 55
Εικόνα 50: Επιτυχής σύνδεση με την βάση δεδομένων μας μέσω VIP. 56
Εικόνα 51 : Μας εξυπηρετεί ο κόμβος mariadbcluster1 56
Εικόνα 52 : Μας εξυπηρετεί ο κόμβος mariadbcluster2 57
Εικόνα 53 : Διάταξη master - slave 57
Εικόνα 54:Διάταξη mariashard1master - mariamstrbckp 58
Εικόνα 55: Αρχείο /etc/my.cnf του mariashard1master 58
Εικόνα 56:Δεδομένα master mariashard1master 58
Εικόνα 57: Αρχείο /etc/my.cnf του mariamstrbckp. 59
Εικόνα 58: Εισαγωγή αρχείου καταγραφής σε mariamstrbckp. 59
Εικόνα 59 : Συνολική Διάταξη spider 60
Εικόνα 60 : Συνολικά VMs που θα χρησιμοποιηθούν στο τρίτο πείραμα 60
Εικόνα 61: Υλοποίηση πίνακα sharding 61
Εικόνα 62: Αναπαράσταση πίνακα sharding μέσω του μηχανισμού spider. 61
Εικόνα 63: Εισαγωγή εγγραφών στο κόμβο spider 62
Εικόνα 64: Εγγραφές στον κόμβο 192.168.56.232 mariashard2 62
Εικόνα 65: Εγγραφές στον κόμβο 192.168.56.233 mariashard3 63
Εικόνα 66: Εγγραφές στον κόμβο mariadbcluster1: 192.168.56.221, mariadbcluster2: 192.168.56.222 63
Εικόνα 67: Εγγραφές στον κόμβο mariamstrbckp: 192.168.56.235 63
Εικόνα 68 : Υλοποίηση 64
Εικόνα 69: Διάταξη spider των πειραματικών διαδικασιών. 65
Εικόνα 70: Διάταξη που θα χρησιμοποιηθεί για την διεξαγωγή μετρήσεων 66
Εικόνα 71: VM που θα χρησιμοποιηθούν στη διεξαγωγή μετρήσεων. 67

Εικόνα 72: Πίνακας sbtest1 του κόμβου spider..68

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Πίνακας 1: Αποτελέσματα Q1 69
Πίνακας 2: Αποτελέσματα Q2 70
Πίνακας 3: Αποτελέσματα Q3 71
Πίνακας 4: Αποτελέσματα Q4 73

ΠΡΟΛΟΓΟΣ

Το αντικείμενο αυτής της εργασίας είναι η μια ολοκληρωμένη λύση για το πρόβλημα μιας επιχείρησης, η οποία διατηρεί μια κεντρικοποιημένη βάση δεδομένων, ενώ διαθέτει ένα σύνολο από πελάτες σε απομακρυσμένες τοποθεσίες.

Αυτή η επιχείρηση τηρεί δεδομένα σε εξυπηρετητές που η ίδια διαθέτει, λόγω του επίπεδου ασφάλειας τους. Μια τέτοια επιχείρηση θα μπορούσε να είναι ο στρατός , όπου τα δεδομένα είναι υψίστης σημασίας για την εθνική ασφάλεια και η τήρησής τους πραγματοποιείται αποκλειστικά από αυτόν.

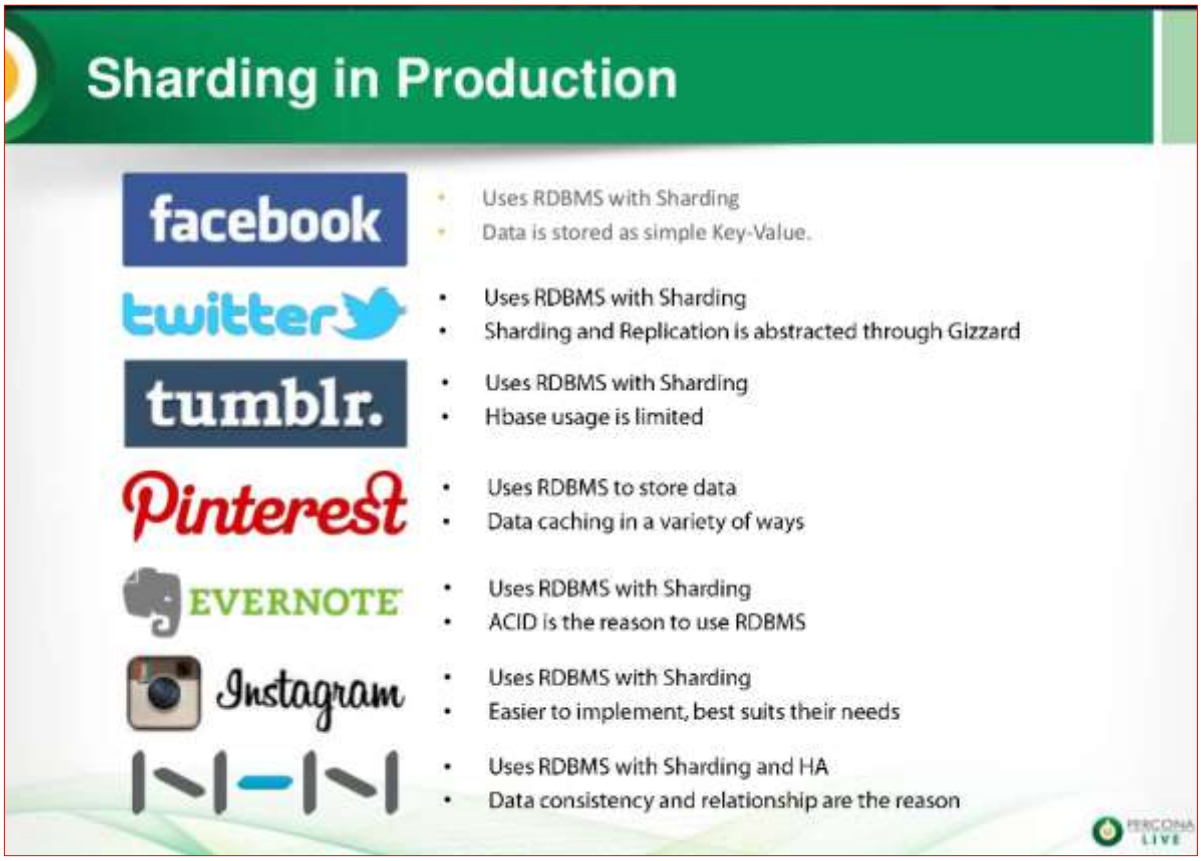
Η προτεινόμενη λύση οδηγεί στον κατακερματισμός της κεντρικοποιημένης βάσης δεδομένων σε ένα σύνολο συστάδων εξυπηρετητών υψηλής διαθεσιμότητας.

1. ΕΙΣΑΓΩΓΗ

1.1 Η εποχή των Μεγάλων Δεδομένων (Big Data)

Η εποχή που διατρέχουμε θα μπορούσε να θεωρηθεί και ως η εποχή των Μεγάλων Δεδομένων. Τα δεδομένα είναι παντού γύρω μας. Η διείσδυση της τεχνολογίας στην καθημερινή μας ζωή έχει σαν αποτέλεσμα την δημιουργία δισεκατομμυρίων δεδομένων από αρχεία καταγραφής ιστού, κινητές συσκευές, αισθητήρες , μέσα και συναλλαγές. Το προηγούμενο έχει ως αποτέλεσμα ο όρος “Μεγάλα Δεδομένα” να χρησιμοποιείται για μια από τις πιο αναδυόμενες τεχνολογίες στον χώρο της πληροφορικής, προκειμένου να περιγράψει την συγκέντρωση και την ανάλυση ιδιαίτερα μεγάλων όγκων δεδομένων και την εξαγωγή συμπερασμάτων, συσχετίσεων και τάσεων που αυτά παρουσιάζουν.

Χαρακτηριστικό παράδειγμα αποτελεί το γεγονός που έλαβε χώρα την 3η Απριλίου 2012 όταν το Instagram προώθησε στο κοινό την έκδοσή τους για Android app και σε λιγότερο από 24 ώρες 1 εκατομμύριο χρήστες εγκατέστησαν την εφαρμογή [1]. Η αρχιτεκτονική που χρησιμοποιήθηκε πίσω από αυτό τον σκοπό ήταν ο κατακερματισμός των δεδομένων μεταξύ διαφορετικών κόμβων δημιουργώντας μια Κατανεμημένη Βάση Δεδομένων [2]. Μια αρχιτεκτονική που χρησιμοποιείται από μεγάλες εταιρείες του διαδικτύου όπως το facebook [3] και το twitter [4] και πολλές ακόμη οι οποίες συγκεντρώνουν δεδομένα από εκατομμύρια χρήστες και η ανάγκη διαχείρισής τους τις οδήγησε στην δημιουργία και ανάπτυξη νέων συστημάτων για αυτόν τον σκοπό, αφού αυτά δεν μπορούσαν να αξιοποιηθούν αποτελεσματικά από τα παραδοσιακά συστήματα βάσεων δεδομένων.



Εικόνα 1: Εταιρείες που χρησιμοποιούν κατακερματισμό των δεδομένων [5].

1.2 Ορισμοί εννοιών

Στο σημείο αυτό θα πρέπει να επεξηγηθούν όροι που χρησιμοποιούνται εκτενώς σε αυτό το κείμενο, οι οποίοι και θα επεξηγηθούν προς διευκόλυνση του αναγνώστη.

Μια Κατανεμημένη Βάση Δεδομένων [6] είναι αυτή που οι πίνακές της κατανέμονται ανάμεσα σε πάνω από έναν υπολογιστές που είναι συνδεδεμένοι σε ένα δίκτυο.

Δύο είναι οι λόγοι για την χρησιμοποίηση κατανεμημένων βάσεων δεδομένων:

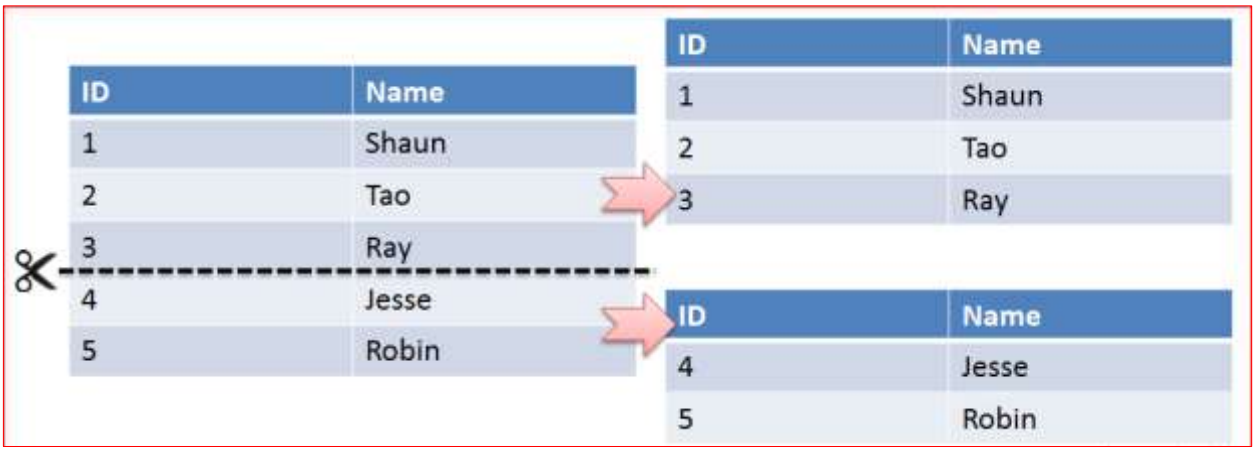
- Ο πρώτος λόγος είναι η απόδοση: συχνά οι πελάτες χρειάζονται κυρίως ένα μικρό τμήμα των δεδομένων μιας εφαρμογής, για παράδειγμα το υποκατάστημα μιας τράπεζας έχει άμεση πρόσβαση στα στοιχεία των δικών του πελατών μόνο. Παίρνοντας τα δεδομένα που χρησιμοποιούνται πιο συχνά και τοποθετώντας τα κοντά στους πελάτες, για παράδειγμα αντιγράφοντάς τα στον εξυπηρετητή ενός τοπικού δικτύου, επιτυγχάνεται αύξηση της αποδοτικότητας.
- Ο δεύτερος λόγος για τον οποίο χρησιμοποιείται η τεχνολογία κατανεμημένων βάσεων δεδομένων προκύπτει από το γεγονός ότι πολλές βάσεις δεδομένων προέκυψαν από συνένωση ανεξάρτητων βάσεων δεδομένων οι οποίες προϋπήρχαν. Η τεχνολογία των κατανεμημένων βάσεων δεδομένων βοήθησε να ενοποιηθούν οι βάσεις δεδομένων.

Τρεις είναι οι τρόποι κατανομής των δεδομένων σε μια κατανεμημένη βάση δεδομένων. Το φόρτωμα , η αναπαραγωγή και ο κατακερματισμός [6]:

- Το φόρτωμα είναι ο απλούστερος τρόπος κατανομής δεδομένων. Περιλαμβάνει την περιοδική αντιγραφή δεδομένων από ένα κεντρικό εξυπηρετητή που έχει όλα τα δεδομένα στους τοπικούς εξυπηρετητές μέσω ενός γρήγορου μέσου μετάδοσης, όπως ένα τοπικό δίκτυο υπολογιστών. Οι τοπικοί εξυπηρετητές παίρνουν μόνο τα δεδομένα που συνήθως χρειάζονται. Πρόκειται για μια πολύ απλή μορφή κατανομής και ο προγραμματισμός της είναι απλός. Ωστόσο, είναι χρήσιμος μόνο σ' εφαρμογές όπου η άμεση ενημέρωση των δεδομένων δεν είναι ζωτικής σημασίας. Ένα παράδειγμα αποτελούν τα δεδομένα μιας τραπεζικής εφαρμογής που ενημερώνει τους λογαριασμούς κατά την διάρκεια της νύχτας. Τα δεδομένα αυτά μπορούν ν' αντιγραφούν στα υποκαταστήματα κατά την έναρξη κάθε εργάσιμης μέρας.
- Ο δεύτερος μηχανισμός κατανομής είναι η αντιγραφή δεδομένων (replication) όπου αντίγραφα της βάσης ή μέρους της αποθηκεύονται σε συγκεκριμένους εξυπηρετητές που βρίσκονται πιο κοντά στους πελάτες που θα ζητήσουν τα δεδομένα. Η λύση αυτή αυξάνει την απόδοση και χρησιμοποιείται όπου απαιτείται συνεχής ενημέρωση των δεδομένων.
- Ο τρίτος τύπος κατανομής είναι ο κατακερματισμός, όπου οι πίνακες μοιράζονται ανάμεσα σε πολλούς υπολογιστές.

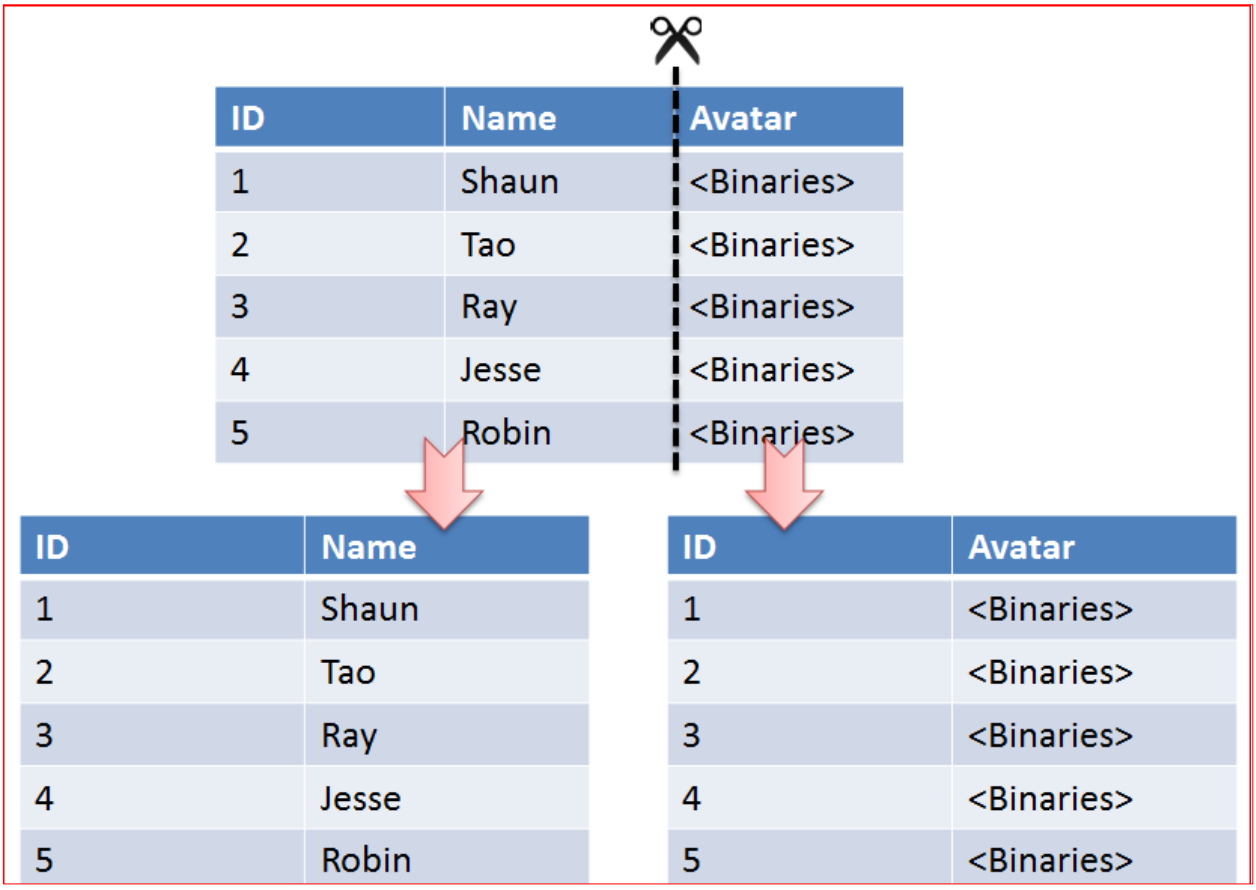
Δύο είναι οι τύποι κατακερματισμού ο οριζόντιος και ο κάθετος κατακερματισμός:

- Ο όρος οριζόντιος κατακερματισμός χρησιμοποιείται για να περιγράψει την διαδικασία διαμερισμού ενός πίνακα σε δύο υπο-πίνακες, που καθένας από αυτούς περιέχει τις όλες τις στήλες του αρχικού πίνακα αλλά μόνο μέρος από τις συνολικές γραμμές του πίνακα. Αυτός ο τύπος κατάτμησης μιας βάσης δεδομένων είναι χρήσιμος όταν ένα υποσύνολο των εγγραφών χρησιμοποιείται συχνά από ένα συγκεκριμένο εξυπηρετητή του συστήματος.



Εικόνα 2: Παράδειγμα οριζόντιου κατακερματισμού

- Ο όρος κατακόρυφος κατακερματισμός αναφέρεται στην διαδικασία διαχωρισμού ενός πίνακα έτσι ώστε κάθε υπο-πίνακας να αποτελείται από δύο πίνακες με μικρότερο αριθμό στηλών σε κάθε πίνακα, σε σχέση με τον αρχικό πίνακα.



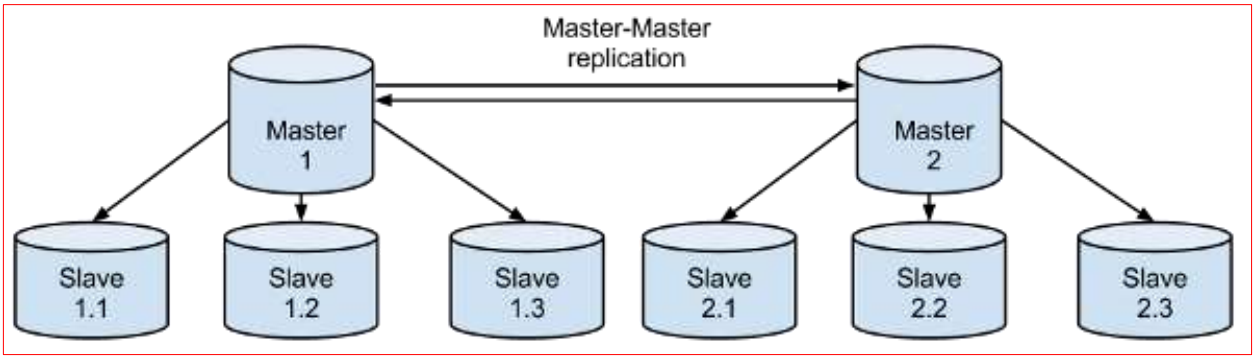
Εικόνα 3: Παράδειγμα κατακόρυφου κατακερματισμού

Αυτός ο τύπος κατακερματισμού χρησιμοποιείται όταν χρειάζεται ένα ειδικό υποσύνολο δεδομένων του πίνακα σε πολύ συγκεκριμένες εφαρμογές. Για παράδειγμα, ένας πίνακας μπορεί να περιέχει στοιχεία για τους υπαλλήλους μιας εταιρείας, με μερικές από τις στήλες του να περιλαμβάνουν στοιχεία σχετικά με τους μισθούς και άλλες σχετικά με την εκπαίδευση και τα προσόντα των υπαλλήλων. Έτσι, το τμήμα

εκπαίδευσης μπορεί να έχει στην κατοχή του έναν πίνακα με τις βασικές λεπτομέρειες αναφορικά με τους υπαλλήλους, όπως το όνομα ή ο τηλεφωνικός τους αριθμός, σε συνδυασμό με στοιχεία για τα προσόντα και την εκπαίδευσή τους, ενώ το τμήμα μισθοδοσίας μπορεί να έχει έναν πίνακα που να περιορίζεται μόνο στην αποθήκευση των βασικών μόνο στοιχείων σε συνδυασμό με τις πληροφορίες για τις πληρωμές και τις ώρες εργασίας.

Δύο είναι οι τύποι αντιγραφής δεδομένων η σύγχρονη και η ασύγχρονη αντιγραφή δεδομένων:

- Ο όρος σύγχρονη αντιγραφή δεδομένων εγγυάται ότι όταν αλλαγές πραγματοποιούνται σε έναν κόμβο μιας συστάδας εξυπηρετητών , πραγματοποιούνται και στους άλλους κόμβους σύγχρονα ή την ίδια στιγμή. Παράδειγμα σύγχρονης αντιγραφής δεδομένων είναι η διάταξη πολλαπλών αφεντών. [7]
- Ο όρος ασύγχρονη αντιγραφή δεδομένων δεν δίνει εγγυήσεις σχετικά με την καθυστέρηση μεταξύ της εφαρμογής των αλλαγών στον αφέντη (master) κόμβο και την διάδοση των αλλαγών στον σκλάβο (slave) κόμβο. [7]



Εικόνα 4: Παράδειγμα διάταξης πολλαπλών αφεντών και σκλάβων.

1.3 Πλεονεκτήματα Κατακερματισμού

Υπάρχουν πολλά πλεονεκτήματα στην προσέγγιση του οριζόντιου κατακερματισμού. Από την στιγμή που οι πίνακες διαιρούνται και κατανέμονται σε πολλαπλούς εξυπηρετητές ο συνολικός αριθμός εγγραφών σε κάθε πίνακα κάθε βάσης δεδομένων μειώνεται. Αυτό μειώνει το μέγεθος του ευρετηρίου της βάσης δεδομένων, το οποίο γενικά βελτιώνει την απόδοση της αναζήτησης. Κάθε τμήμα της βάση που έχει κατανεμηθεί , το οποίο αποκαλείται θραύσμα (shard), μπορεί να τοποθετηθεί σε διαφορετικό υλισμικό (hardware) και πολλαπλά θραύσματα μπορούν να τοποθετηθούν σε πολλαπλούς εξυπηρετητές. Αυτό επιτρέπει την κατανομή της βάσης δεδομένων σε ένα μεγάλο αριθμό εξυπηρετητών, βελτιώνοντας σε μεγάλο βαθμό την απόδοση. Επιπλέον η κατάτμηση της βάσης δεδομένων σε θραύσματα έχει πραγματοποιηθεί με κριτήριο την τοποθεσία των χρηστών (πχ πελάτες_ευρώπης ,πελάτες_αμερικής) ,τότε είναι δυνατό να επηρεάσει την συνολική απόδοση καθώς οι χρήστες θα κάνουν ερωτήματα μόνο στα σχετικά θραύσματα.

1.4 Προβλήματα Κατακερματισμού

Υπάρχουν ορισμένα προβλήματα με τις κατανεμημένες βάσεις δεδομένων που χρειάζονται διαλεύκανση τόσο από την πλευρά του διακομιστή όσο και από την πλευρά του πελάτη. Τα προβλήματα αυτά είναι απλά μια άλλη εκδοχή των γενικότερων προβλημάτων που πρέπει να επιλυθούν σε μια οποιαδήποτε κατανεμημένη εφαρμογή.

- Αν υπάρχουν πολλά αντίγραφα των δεδομένων σε πολλούς διακομιστές θα πρέπει τα διπλότυπα δεδομένα να είναι ίδια σε όλους τους διακομιστές. Θα πρέπει δηλαδή όταν γίνεται μια αλλαγή να ενημερώνονται όλοι οι διακομιστές που εμπλέκονται, χωρίς, όμως, να διακινδυνεύεται η απόδοση: οι πελάτες δεν θα πρέπει να περιμένουν μέχρις ότου η βάση συγχρονιστεί με όλα της τα αντίγραφα.
- Θα πρέπει να εξασφαλιστεί ότι η ταυτόχρονη πρόσβαση σε μία κατανεμημένη βάση δεδομένων γίνεται με ασφαλή για την λογική ακεραιότητα των δεδομένων τρόπο. Για παράδειγμα, ένας χρήστης μπορεί να διατηρεί στην μνήμη ορισμένα δεδομένα, αλλά, την ίδια στιγμή, ένας άλλος χρήστης μπορεί να γράφει και να διαβάζει τα δεδομένα πριν ο πρώτος χρήστης στείλει τα αποτελέσματά του πίσω στην βάση δεδομένων. Αυτό θα έχει ως αποτέλεσμα να μην είναι σωστά τα στοιχεία που περιέχει η βάση δεδομένων. Υπάρχουν κάποιες πολύπλοκες τεχνικές για την αντιμετώπιση αυτού του προβλήματος.
- Σ' ένα κατανεμημένο περιβάλλον με πολλούς υπολογιστές, η ασφάλεια είναι ένα πολύ σημαντικό ζήτημα.
- Η αξιοπιστία αποτελεί επίσης πρόβλημα. Για παράδειγμα, οι δοσοληψίες σε ένα κατανεμημένο σύστημα βάσεων δεδομένων μπορεί να απαιτούν μια πολύ μεγάλη, ιδιαίτερα πολύπλοκη και ευαίσθητη σε λάθη διαδικασία. Πρέπει πάντα να διατηρείται η αξιοπιστία και αυτό μπορεί να αποδειχτεί ιδιαίτερα δύσκολο σε ένα περιβάλλον κατανεμημένων βάσεων δεδομένων.
- Ένα από τα προβλήματα των κατανεμημένων συστημάτων είναι ότι τα ρολόγια των υπολογιστών που συνδέονται με το σύστημα μπορεί να μην είναι συγχρονισμένα μεταξύ τους. Όταν ορισμένες συναλλαγές πρέπει να εκτελεστούν με κάποια χρονική σειρά τότε ο συγχρονισμός είναι αναγκαίος και μπορεί να είναι αρκετά δύσκολος

Τα προβλήματα αυτά είναι δύσκολο να λυθούν. Ωστόσο, τα καλά νέα αν είστε προγραμματιστές είναι ότι συνήθως δεν χρειάζεται να ανησυχείτε για τέτοιου είδους προβλήματα μιας και οι διακομιστές και τα λειτουργικά συστήματα που χρησιμοποιείτε συνήθως δίνουν την λύση γι' αυτά χωρίς να χρειαστεί να καταλαβαίνετε πως τα ζητήματα αυτά λύνονται. Αν είστε σχεδιαστής κατανεμημένων συστημάτων, το μόνο που πρέπει να σας απασχολεί είναι να διαλέξετε εργαλεία λογισμικού για την ανάπτυξη του συστήματος σας, τα οποία θα καλύπτουν προβλήματα των ειδών που αναφέραμε και που μπορούν να παρουσιαστούν στην εφαρμογή σας ώστε να έχετε μια αξιόπιστη εφαρμογή.[6]

1.5 Ιδιότητες ACID

Προκειμένου να διασφαλιστεί η ακεραιότητα των δεδομένων τα περισσότερα από τα κλασικά συστήματα βάσεων δεδομένων βασίζονται σε συναλλαγές. Αυτό εξασφαλίζει συνοχή των δεδομένων σε όλες τις περιπτώσεις της διαχείρισης τους. Το σύνολο των ιδιοτήτων το οποίο εγγυάται ότι οι συναλλαγές στη βάση δεδομένων λειτουργούν αξιόπιστα είναι γνωστό με το ακρωνύμιο ACID (Atomicity , Consistency, Isolation, Durablity – Ατομικότητα, Συνεκτικότητα, Απομόνωση , Μονιμότητα)

Στα τέλη του 1970 ο Jim Gray όρισε τις ιδιότητες ενός συστήματος αξιόπιστων συναλλαγών και ανάπτυξε τεχνολογίες που να το επιτυγχάνουν. Το 1983 ο Andreas Reuter και ο Theo Haerder χρησιμοποίησαν το ακρωνύμιο ACID για να περιγράψουν αυτές τις ιδιότητες.[8]

Ατομικότητα (Atomicity) [9]

Η Ατομικότητα απαιτεί η τροποποίηση που θα γίνει στην ΒΔ να τηρεί τον κανόνα όλα ή τίποτα. Κάθε συναλλαγή, η οποία είναι ατομική, ονομάζεται έτσι επειδή αν ένα μέρος της αποτύχει, αποτυγχάνει όλη η συναλλαγή και η ΒΔ μένει όπως ήταν πριν εκτελεστεί

η συναλλαγή. Είναι πολύ κρίσιμο το ΣΔΒΔ να διατηρεί την ατομικότητα κάθε συναλλαγής ανεξάρτητα από το είδος της εφαρμογής, του ΣΔΒΔ, του λειτουργικού συστήματος ή αστοχίες του υλικού.

Μια ατομική συναλλαγή δεν μπορεί να υποδιαιρεθεί, και πρέπει να επεξεργάζεται ολόκληρη ή καθόλου. Η Ατομικότητα σημαίνει ότι οι χρήστες είναι απαλλαγμένοι από τον φόβο μη ολοκληρωμένων συναλλαγών.

Οι συναλλαγές μπορούν να αποτύχουν για πολλούς και διάφορους λόγους:

- Αποτυχία Υλικού: Ένας δίσκος μπορεί να χαλάσει, αποτρέποντας μερικές αλλαγές στη ΒΔ από το να πραγματοποιηθούν.
- Αποτυχία Συστήματος: Ο χρήστης μπορεί να χάσει την σύνδεση του με το σύστημα πριν ολοκληρώσει τις εργασίες του.
- Αποτυχία ΒΔ: Η μνήμη της ΒΔ τελειώνει και δεν μπορεί να καταχωρήσει άλλα δεδομένα.
- Αποτυχία Εφαρμογής: Η εφαρμογή η οποία χειρίζεται την βάση δεδομένων προσπαθεί να βάλει δεδομένα τα οποία παραβιάζουν τους κανόνες που έχει επιβάλλει η ΒΔ, πχ απόπειρα δημιουργίας νέου λογαριασμού χωρίς αριθμό λογαριασμού.

Συνέπεια (Consistency)

Η ιδιότητα της Συνέπειας διασφαλίζει ότι η ΒΔ διατηρείται σε μια συνεπή κατάσταση, συγκεκριμένα λέει ότι κάθε συναλλαγή θα οδηγεί την βάση δεδομένων από την μια συνεπή κατάσταση στη άλλη.

Η ιδιότητα της συνέπειας δεν λέει πως ένα ΣΔΒΔ πρέπει να χειρίζεται μια ασυνέπεια αλλά λέει πώς να είναι καθαρή η ΒΔ στο τέλος μιας συναλλαγής. Αν για κάποιο λόγο μια συναλλαγή παραβιάζει την συνέπεια της ΒΔ όλη η συναλλαγή ακυρώνεται (επανέρχεται στην κατάσταση πριν εκτελεστεί η συναλλαγή) ή το ΣΔΒΔ κάνει μερικές ενέργειες για να είναι η ΒΔ και πάλι συνεπής. Έτσι αν ένα σχήμα ΒΔ λέει πως ένα πεδίο είναι μόνο για ακέραιους αριθμούς τότε το ΣΔΒΔ μπορεί είτε να απορρίψει απόπειρες για είσοδο δεκαδικών αριθμών είτε να τους στρογγυλοποιήσει. Και οι δυο αυτές ενέργειες διατηρούν την συνέπεια.

Ο κανόνας συνέπειας ισχύει μόνο σε κανόνες ακεραιότητας οι οποίοι είναι εντός της εμβέλειας του αντικειμένου. Έτσι, αν ένα ΣΔΒΔ επιτρέπει σε κάποια πεδία να είναι αναφορές για άλλη εγγραφή τότε η συνέπεια προϋποθέτει ότι το ΣΔΒΔ πρέπει να ενισχύσει την περιφερειακή ακεραιότητα (referential integrity).Περιφερειακή ακεραιότητα: είναι μια ιδιότητα δεδομένων η οποία όταν ισχύει απαιτεί κάθε τιμή από ένα χαρακτηριστικό (στήλη στην ΒΔ) μιας σχέσης (πίνακας) να υπάρχει σαν τιμή κάποιου άλλου χαρακτηριστικού σε μια διαφορετική (ή ίδια) σχέση.) Όταν τελειώσει οποιαδήποτε συναλλαγή οποιαδήποτε αναφορά στην ΒΔ πρέπει να είναι έγκυρη και να ισχύει. Αν μια συναλλαγή αποπειραθεί να διαγράψει μια εγγραφή η οποία αναφέρεται από κάπου αλλού τότε ένας από τους ακόλουθους μηχανισμούς θα διατηρήσουν την συνέπεια:

- Η συναλλαγή να ακυρωθεί (abort) και να γυρίσει η ΒΔ στην προηγούμενη συνεπή κατάσταση.
- Να διαγράφουν όλα τα αρχεία τα οποία ανέφεραν αυτή τη διαγραμμένη εγγραφή (το οποίο είναι γνωστό σαν cascade delete).
- Να μηδενίσει (τιμές null) όλα τα σχετικά πεδία σε όλες τις εγγραφές οι οποίες είχαν σχέση με την διαγραμμένη εγγραφή.

Αυτά είναι παραδείγματα Διάδοσης Περιορισμών, μερικά συστήματα ΒΔ επιτρέπουν στους σχεδιαστές της βάσης να καθορίζουν ποιες επιλογές πρέπει να υπάρχουν όταν δημιουργείται ένα σχήμα για μια ΒΔ.

Οι δημιουργοί εφαρμογών είναι υπεύθυνοι για διατήρηση τις συνέπειας όσο αφορά τις εφαρμογές τους, επιπλέον από τους προσφερόμενους κανόνες του ΣΔΒΔ. Έτσι, αν ένας χρήστης κάνει μια ανάληψη κεφαλαίου και το νέο υπόλοιπο είναι χαμηλότερο από το επιτρεπόμενο όριο τότε όσο αφορά ο ΣΔΒΔ η βάση είναι σε συνεπή κατάσταση αν και ο κανόνας που το ελέγχει αυτό έχει παραβιαστεί.

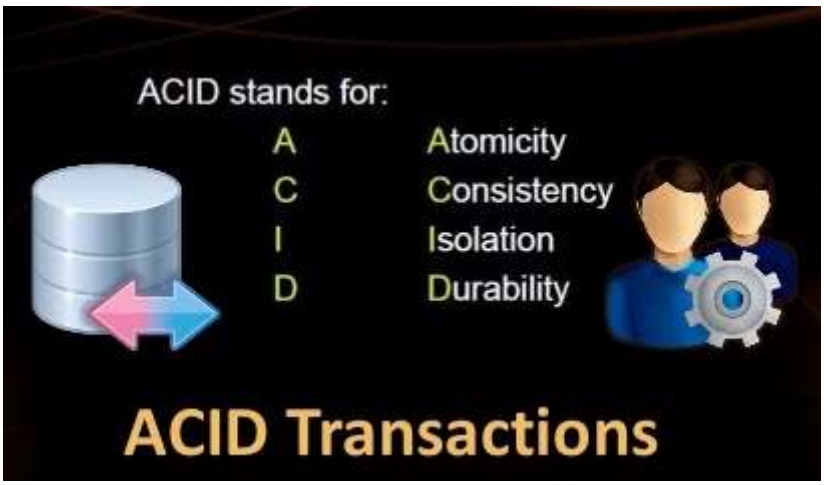
Απομόνωση (Isolation)

Η Απομόνωση αναφέρεται στην απαίτηση ότι όλες οι ενέργειες δεν μπορούν να έχουν πρόσβαση ή να δουν δεδομένα τα οποία τροποποιούνται εκείνη την στιγμή από μια συναλλαγή η οποία δεν έχει ακόμα ολοκληρωθεί. Κάθε συναλλαγή δεν πρέπει να ξέρει αν υπάρχουν άλλες συναλλαγές που εκτελούνται ταυτόχρονα, αλλά να περιμένουν την ολοκλήρωση μιας συναλλαγής ώστε να δουν/τροποποιήσουν τα δεδομένα τα οποία χρειάζεται και η άλλη συναλλαγή.

Μονιμότητα (Durability)

Η Μονιμότητα εγγυάται στον χρήστη το ΣΔΒΔ ότι αν τελειώσει μια συναλλαγή επιτυχώς τότε τα αποτελέσματα της δεν θα χαθούν. Οι αλλαγές που έχει κάνει η συναλλαγή δεν θα χαθούν έστω και να κρασάρει το σύστημα και ότι όλες οι προϋποθέσεις ακεραιότητας ισχύουν, έτσι ώστε το ΣΔΒΔ δεν θα χρειαστεί αν ακυρώσει αυτή τη συναλλαγή. Πολλά ΣΔΒΔ επιτυγχάνουν μονιμότητα με το να τηρούν ένα αρχείο συναλλαγών (log) το οποίο μπορεί να χρησιμοποιηθεί και να επαναφέρει την κατάσταση του συστήματος ακριβώς όπως ήταν πριν το κρασάρισμα. Μια συναλλαγή αποκαλείται commit μόνο όταν καταχωρηθεί σε αυτό το αρχείο.

Η Μονιμότητα δεν προϋποθέτει μόνιμη κατάσταση της ΒΔ. Μια υποδεέστερη συναλλαγή μπορεί να αλλάξει δεδομένα τα οποία έχουν αλλαχτεί από μια κυριότερη συναλλαγή χωρίς να υπάρχει παραβίαση της αρχής της μονιμότητας.



Εικόνα 5: ACID ιδιότητες [10]

Κλείδωμα VS Multiversioning

Οι περισσότερες Βάσεις βασίζονται στα κλειδώματα για να παρέχουν τις ιδιότητες ACID. Το κλείδωμα σημαίνει ότι η συναλλαγή μαρκάρει τα δεδομένα πριν τα χρησιμοποιήσει έτσι ώστε το ΣΔΒΔ να μην επιτρέπει σε άλλες συναλλαγές να το τροποποιήσουν μέχρις ότου η συναλλαγή αυτή τελειώσει μαζί τους (με τα δεδομένα). Το

κλείδωμα πρέπει πάντα να αποκτάται πριν την επεξεργασία των δεδομένων ακόμη και δεδομένα τα οποία διαβάζονται μόνο. Επουσιώδης συναλλαγές απαιτούν ένα μεγάλο αριθμό κλειδωμάτων το οποίο οδηγεί σε σημαντική χρήση πόρων και μπλοκάρισμα άλλων συναλλαγών. Για παράδειγμα, αν ο χρήστης Α τρέχει μια συναλλαγή η οποία πρέπει να διαβάσει μια σειρά δεδομένων τότε ο χρήστης Β πρέπει να περιμένει μέχρι να τελειώσει η συναλλαγή του Α. Συχνά εφαρμόζεται κλείδωμα δυο φάσεων για να εγγυηθεί πλήρης απομόνωση.

Μια εναλλακτική λύση κλειδώματος είναι ο έλεγχος ταυτοχρονισμού multiversion με τον οποίο η ΒΔ παρέχει σε κάθε συναλλαγή read προτεραιότητα και μη τροποποιημένη έκδοση των δεδομένων, τα οποία τροποποιούνται από άλλη ενεργή συναλλαγή που θέλει να τα τροποποιήσει. Αυτό επιτρέπει στις συναλλαγές που θέλουν να κάνουν read να το κάνουν χωρίς κλείδωμα. Για παράδειγμα, οι write συναλλαγές δεν μπλοκάρουν τις read συναλλαγές και οι συναλλαγές read δεν μπλοκάρουν τις συναλλαγές write. Πηγαίνοντας στο προηγούμενο παράδειγμα όταν η συναλλαγή του χρήστη Α απαιτήσει δεδομένα τα οποία τροποποιεί εκείνη την στιγμή η συναλλαγή του Β τότε η ΒΔ παρέχει στον Α τα δεδομένα τα οποία υπήρχαν πριν η συναλλαγή του χρήστη Β αρχίσει την τροποποίηση. Ο χρήστης Α έχει μια συνεπή όψη της ΒΔ ακόμα και αν άλλοι χρήστες την τροποποιούν. Για μια τέτοια υλοποίηση δεν είναι τόσο αυστηρή η ιδιότητα της απομόνωσης, η οποία ονομάζεται snapshot isolation.

Κατανεμημένες Συναλλαγές

Εξασφαλίζοντας τις ιδιότητες ACID σε μια κατανεμημένη συναλλαγή σε μια κατανεμημένη ΒΔ όπου δεν υπάρχει ένας κόμβος για όλα τα δεδομένα μια συναλλαγή παρουσιάζει πολλές επιπλοκές. Το δίκτυο μπορεί να αποσυνδεθεί ή ένας κόμβος να ολοκληρωθεί επιτυχώς και μετά να πρέπει να επιστρέψει στην προηγούμενη κατάσταση επειδή απέτυχε ένας άλλος κόμβος. Το πρωτόκολλο two-phase commit (2PC) (το οποίο είναι διαφορετικό από το πρωτόκολλο κλειδώματος δύο φάσεων) παρέχει ατομικότητα για κατανεμημένες συναλλαγές ώστε να εξασφαλίσει ότι κάθε μέρος της συναλλαγής να συμφωνεί αν η συναλλαγή πρέπει να γίνει commit ή abort. Συνοπτικά, στην πρώτη φάση ένας κόμβος (ο συντονιστής) διερευνά τους άλλους κόμβους και μόνο όταν απαντήσουν ότι είναι έτοιμοι επικυρώνει την συναλλαγή στη δεύτερη φάση.

Καταλήγοντας η απομάκρυνση από τα συστήματα που έχουν ιδιότητες ACID έχει αποδειχθεί ότι δημιουργεί διάφορα είδη προβλημάτων. Συγκρούσεις προκύπτουν μεταξύ των διαφόρων πτυχών της υψηλής διαθεσιμότητας σε κατανεμημένα συστήματα, οι οποίες δεν μπορούν να επιλυθούν πλήρως. Την κατάσταση αυτή περιγράφει το θεώρημα CAP που ακολουθεί.

1.6 Οι κανόνες του Date

Ο Chris Date, που ήταν ένας από τους πρωτοπόρους της σχεσιακής τεχνολογίας, επινόησε 12 κανόνες σύμφωνα με τους οποίους θα πρέπει να κρίνεται η αποτελεσματικότητα μιας τεχνολογίας κατανεμημένων βάσεων δεδομένων.[11]

- Τοπική αυτονομία . Κάθε διακομιστής μπορεί να δρα ως ανεξάρτητο, αυτόνομο και κεντρικοποιημένο ΣΔΒΔ. Θα πρέπει είναι υπεύθυνο για την ασφάλεια, έλεγχο ταυτότητας, τήρηση αντιγράφων ασφαλείας και ανάκτηση δεδομένων
- Ανεξαρτησία από τον κεντρικό διακομιστή: Οι διακομιστές θα πρέπει να είναι αυτόνομοι από τον κεντρικό διακομιστή.
- Συνεχής λειτουργία. Ένα σύστημα κατανεμημένης βάσης δεδομένων θα πρέπει να λειτουργεί διαρκώς είτε παρουσιάζεται αποτυχία σε κάποιο κόμβο είτε επέκταση του δικτύου.
- Διαφάνεια Τοπικότητας: Ο χρήστης δεν θα πρέπει να γνωρίζει που βρίσκονται τα δεδομένα για να αποκτήσει πρόσβαση σε αυτά.

- Ανεξαρτησία Κατάτμησης. Η κατάτμηση της βάσης δεδομένων είναι διαφανής στο χρήστη, ο οποίος βλέπει μία μόνο λογική βάση, δεν χρειάζεται να ξέρει ότι η βάση είναι κατανεμημένη.
- Ανεξαρτησία αντιγραφής δεδομένων. Ο προγραμματιστής ή ο χρήστης δεν θα πρέπει να γνωρίζει ότι υπάρχουν πολλά αντίγραφα μίας βάσης δεδομένων ή τμημάτων της.
- Διαδικασία επεξεργασία κατανεμημένων ερωτημάτων. Η μηχανή επεξεργασίας ερωτημάτων ενός διακομιστή βάσης δεδομένων θα πρέπει να είναι ενήμερη για την κατανομή δεδομένων και να είναι σε θέση ν' αποφασίζει για τον τρόπο ανάκτησης των δεδομένων ανάλογα με την τοποθεσία στην οποία βρίσκονται
- Κατανεμημένη επεξεργασία συναλλαγών. Μια συναλλαγή μπορεί να αναβαθμίσει τα δεδομένα σε διαφορετικούς διακομιστές και η συναλλαγή να εκτελείται με διαφάνεια.
- Ανεξαρτησία υλισμικού: Το σύστημα πρέπει να εκτελείται σε κάθε μια πλατφόρμα.
- Ανεξαρτησία λειτουργικών συστημάτων. Ένα σύστημα δεν θα πρέπει να επηρεάζεται από το λειτουργικό σύστημα που ενίοτε χρησιμοποιείται.
- .Ανεξαρτησία Δικτύου: Το σύστημα πρέπει να εκτελείται σε κάθε πλατφόρμα δικτύου.
- Ανεξαρτησία βάσης δεδομένων: Το σύστημα πρέπει να υποστηρίζει διαφορετικά μοντέλα βάσεων δεδομένων.

1.7 Θεώρημα CAP

Το 2000 στο συμπόσιο με θέμα τις αρχές των κατανεμημένων συστημάτων ο Eric Brewer εξέφρασε την άποψη ότι κάθε διαμοιρασμένο σύστημα μπορεί να έχει δύο από τις τρεις εξής βασικές ιδιότητες: [12]

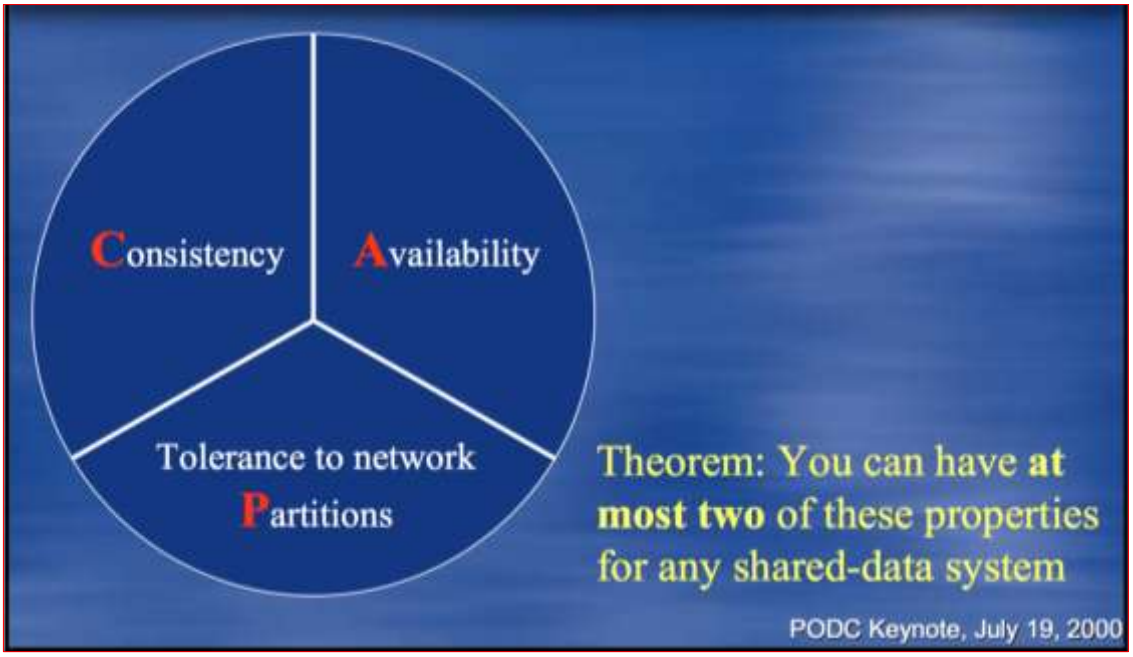
- Συνέπεια (Consistency)
- Διαθεσιμότητα (Availabilty)
- Ανοχή στις κατατμήσεις που οφείλονται στο δίκτυο (Tolerance to network Partitions)

Το 2002 οι N. Lynch και S. Gilbert έδωσαν μία απόδειξη για την εκτίμηση του Brewer οπότε και αυτή καθιερώθηκε σαν θεώρημα CAP. [14]

Αναλυτικότερα, η ιδιότητα της συνέπειας έχει την έννοια, ότι κάθε ερώτημα που γίνεται στη βάση, θα διαβάζει τα τελευταία δεδομένα που καταχωρήθηκαν σε αυτήν και όχι παλαιότερα. Για την απόδειξη του CAP θεωρήματος χρησιμοποιήθηκε η έννοια της ατομικότητας. Δηλαδή, το σύστημα θα πρέπει να συμπεριφέρεται σαν να αποτελείτο από έναν κόμβο στον οποίο εγγραφές και αναγνώσεις εξυπηρετούνται με τη σειρά που πραγματοποιήθηκαν. Η ιδιότητα της διαθεσιμότητας έχει την έννοια ότι κάθε αίτημα που γίνεται στη βάση θα πρέπει να λαμβάνει απάντηση, είτε αυτό έχει επιτύχει, είτε έχει αποτύχει. Τέλος η ανοχή στις κατατμήσεις σημαίνει ότι σε περίπτωση που κάποιο μέρος του συστήματος αποκοπεί από το υπόλοιπο λόγω βλάβης στο δίκτυο (ή και βλάβης σε κάποια μηχανήματα), τότε το σύστημα θα πρέπει να είναι σε θέση να συνεχίσει να εξυπηρετεί αιτήματα και να λειτουργεί.

Τα επόμενα χρόνια, που αναπτύχθηκαν σε πολύ μεγαλύτερο βαθμό τέτοιου είδους συστήματα, έγινε κοινώς αποδεκτό, ότι οι δημιουργοί των συστημάτων θα έπρεπε όντως να επιλέξουν ποιες δύο από τις τρεις ιδιότητες θα υποστήριζε αποτελεσματικά το σύστημά τους σε περιπτώσεις που κάτι δεν πήγαινε καλά. Η ανοχή στις κατατμήσεις ήταν κάτι που από τη φύση τους έχουν ανάγκη τα κατανεμημένα συστήματα, επομένως έπρεπε να επιλέξουν σύμφωνα με τις ανάγκες για τις οποίες προοριζόταν το σύστημα, για το ποια από τις δύο ιδιότητες Διαθεσιμότητα - Συνέπεια θα υποστήριζαν κατά

προτεραιότητα (και φυσικά όχι κατ' αποκλειστικότητα). Παρά το γεγονός ότι το θεώρημα CAP ορίζει ότι πρέπει να διαλέξει κανείς ανάμεσα σε διαθεσιμότητα και σε συνέπεια αν θέλει να έχει ανοχή στις κατατμήσεις, αυτό δεν σημαίνει ότι η επιλογή της διαθεσιμότητας θα έχει σαν αποτέλεσμα να μην εξασφαλίζεται καμία συνοχή. Αυτό που συμβαίνει είναι ότι χρησιμοποιούνται μηχανισμοί οι οποίοι παρέχουν συνέπεια τελικώς (eventual consistency), δηλαδή τα δεδομένα που ενημερώνονται σε κάποιο κόμβο θα ενημερωθούν σταδιακά και στους υπόλοιπους κόμβους μέσω των μηχανισμών αυτών . Αυτή η προσέγγιση μπορεί να μην παρέχει εγγυήσεις ότι τα δεδομένα που διαβάζονται σε κάποια χρονική στιγμή είναι τα πιο ενημερωμένα, αλλά τουλάχιστον κάνει το μεγαλύτερο μέρος της βάσης να είναι σε συνεπή κατάσταση.



Εικόνα 6:Το θεώρημα CAP [13]

1.8 Σκοπός

Στην διάταξη που θα παρουσιάσουμε θα μελετήσουμε σε πρώτο επίπεδο μια συστάδα εξυπηρετητών Γαλέρα (Galera Cluster) της MariaDB, που θα υποστηρίζει τόσο την σύγχρονη όσο και την ασύγχρονη αντιγραφή δεδομένων

Σε δεύτερο επίπεδο θα μελετηθεί ο κατακερματισμός των δεδομένων μεταξύ διαφορετικών συστάδων εξυπηρετητών για να παραχθεί και η επιθυμητή αρχιτεκτονική.

Ο σκοπός ο οποίος θα επιτευθεί θα είναι υψηλή διαθεσιμότητα (high availability) και κατανομή φορτίου (load balancing) σε κάθε μια συστάδα εξυπηρετητών, με τρόπο απολύτως διαφανές προς τον χρήστη, μέσω του μηχανισμού Υψηλής Διαθεσιμότητας Εξυπηρετητή Μεσολάβησης (HAProxy).

Στο δεύτερο επίπεδο θα αποδειχθεί η κλιμακωσιμότητα και βελτίωση της απόδοσης στο σύνολο των συστάδων εξυπηρετητών, το οποίο θα προέλθει από τον κατακερματισμό των δεδομένων μεταξύ των συστάδων, που προσφέρει ο μηχανισμός αποθήκευσης δεδομένων (storage engine) Spider.

.

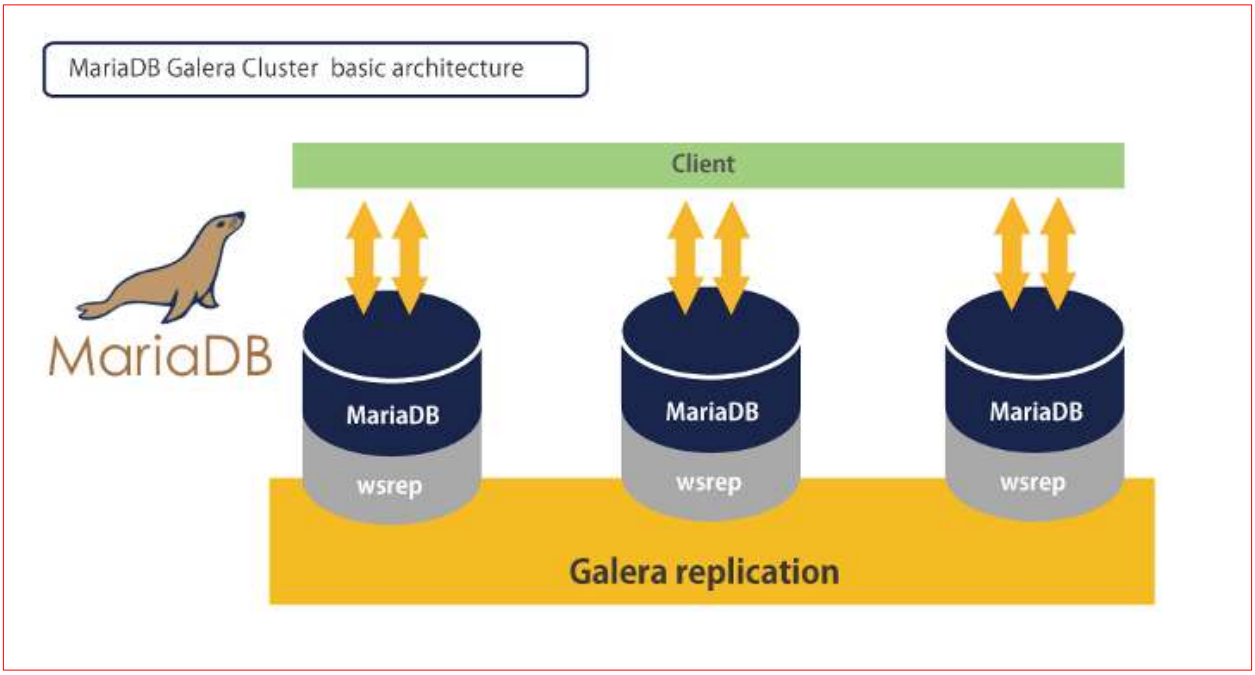
2. ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΩΝ

2.1 Συστάδα εξυπηρετητών Γαλέρα : Εισαγωγή

Η συστάδα εξυπηρετητών Γαλέρα είναι μια σύγχρονης αντιγραφής δεδομένων συστάδα εξυπηρετητών πολλαπλών αφεντών της MariaDB. Είναι διαθέσιμη σε Linux και υποστηρίζει μηχανισμούς αποθήκευσης όπως XtraDB/InnoDB, ενώ σε πειραματικό επίπεδο είναι η MyISAM . Ξεκινώντας από την έκδοση MariaDB 10.1 , η wsrep API για την συστάδα εξυπηρετητών γαλέρα περιέχεται στην συνολική εγκατάσταση. Αυτή είναι διαθέσιμη ας ξεχωριστή εγκατάσταση για της εκδόσεις MariaDB 10.0 και MariaDB 5.5. [15]

Η wserp API καθορίζει ένα σετ από επανακλήσεις (callbacks) και κλήσεις αντιγραφής βιβλιοθήκης οι οποίες είναι απαραίτητες για να εφαρμοστεί σύγχρονη αντιγραφή δεδομένων σε βάσεις δεδομένων συναλλαγών (transactional databases) καθώς και παρόμοιες εφαρμογές. Αν και ο κύριος σκοπός αυτής της διεπαφής (interface) είναι μια πολλαπλών αφεντών αντιγραφή δεδομένων με πιστοποίηση , είναι ισότιμα κατάλληλα για ασύγχρονη και σύγχρονη σε αντιγραφή δεδομένων αφέντη / σκλάβο αντιγραφή. [16]

Οι επανακλήσεις εφαρμογής επιτρέπουν διαχείριση συναλλαγών για εγγραφές, τόσο εφαρμόζοντας αυτές τις εγγραφές στον αποστολέα, όσο και εφαρμόζοντας τες στον αποδέκτη.



Εικόνα 7: Βασική αρχιτεκτονική συστάδας εξυπηρετητών MariaDB.

2.1.1 Χαρακτηριστικά συστάδας εξυπηρετητών Γαλέρα

Τα χαρακτηριστικά της συστάδα εξυπηρετητών είναι : [15]

- Η σύγχρονη αντιγραφή δεδομένων.
- Ενεργή – Ενεργή πολλαπλών αφεντών τοπολογία.
- Η ανάγνωση και εγγραφή σε κάθε κόμβο της συστάδας εξυπηρετητών
- Ο αυτόματος έλεγχος διαχείρισης μέλους (membership control), οι αποτυγχάνοντες κόμβοι βγαίνουν από την συστάδα εξυπηρετητών.
- Η αυτόματη εισαγωγή κόμβου στην συστάδα εξυπηρετητών.

- Πραγματική παράλληλη αντιγραφή δεδομένων, σε επίπεδο γραμμής.
- Άμεσες συνδέσεις πελάτη, διαφανής αίσθηση της MariaDB.

2.1.2 Λειτουργία Συστάδας εξυπηρετητών Γαλέρα

Η πρωτεύουσα λειτουργία της συστάδα εξυπηρετητών Γαλέρα είναι η συνοχή των δεδομένων. Οι συναλλαγές πραγματοποιούνται είτε σε όλους τους κόμβους είτε απορρίπτονται. Έτσι οι βάσεις παραμένουν συγχρονισμένες, με την προϋπόθεση ότι ρυθμίστηκαν και συγχρονίστηκαν κατάλληλα κατά την εκκίνηση της συστάδας. [17]

Όταν οι κόμβοι εκκινούν, επιδιώκουν να αποκτήσουν σύνδεση δικτύου με τους άλλους κόμβους στην συστάδα εξυπηρετητών. Για κάθε έναν κόμβο που βρίσκουν, ελέγχουν άμα ανήκει ή όχι στο τμήμα του πρωτεύοντος τμήματος (Primary Component). Όταν βρουν το πρωτεύων τμήμα, κάνουν αίτηση για μεταφορά κατάστασης για να συγχρονίσουν την τοπική τους βάση με αυτή της συστάδας.

Όταν ξεκινά η συστάδα δεν υπάρχει πρωτεύων τμήμα, για να αρχικοποιηθεί θα πρέπει να καθορίσουμε έναν κόμβο να το κάνει, με το να είναι ο πρώτος κόμβος που θα εισαχθεί στην συστάδα και θα γίνει λειτουργικός.

Κάθε κόμβος που εισάγεται έπειτα στην συστάδα ,συνδέεται με τα υπόλοιπα μέλη της συστάδας. Όταν όλοι οι κόμβοι συμφωνούν για την κατάσταση των μελών, πραγματοποιούν έναρξη ανταλλαγής κατάστασης (state exchange). Σε αυτήν την κατάσταση ο κόμβος ελέγχει την κατάσταση της συστάδας. Αν η κατάσταση του κόμβου διαφέρει από αυτή της συστάδας το οποίο και συμβαίνει επί το πλείστον , ο νέος κόμβος ζητάει μεταφορά κατάστασης στιγμιότυπου (snapshot state transfer -SST) και την εγκαθιστά στην τοπική του βάση. Μετά από αυτή την εργασία ο νέος κόμβος είναι δυνατό να χρησιμοποιηθεί.

Όταν η συστάδα εξυπηρετητών είναι λειτουργική, η Γαλέρα μεταφέρει όλες τις συναλλαγές (transactions) σε όλους τους κόμβους της συστάδας και εφαρμόζει τις αλλαγές ή τις αποκλείει σε περίπτωση σύγκρουσης δεδομένων.

Αν κάποια σύνδεση απολεσθεί , όλοι οι κόμβοι ελέγχουν ότι διαθέτουν την συγκατάθεση της πλειοψηφίας (majority consensus) και σταματάνε να ικανοποιούν αιτήσεις άμα δεν την διαθέτουν.

Όταν ένας κόμβος που είχε διακόψει την σύνδεση επιστρέφει στην συστάδα, λαμβάνει όλες τις αλλαγές , πραγματοποιώντας αύξουσα μεταφορά κατάστασης (incremental state transfer IST) και σαν αποτέλεσμα κάνει το περιεχόμενο του πρόσφατο. Άμα το προηγούμενο δεν είναι δυνατό, πραγματοποιεί μια ολική μεταφορά SST. Στην περίπτωση που δεν πραγματοποιηθεί κάτι από τα προηγούμενα δεν γίνεται αποδεκτός από την συστάδα και παραμένει μη λειτουργικός.

Για τους προηγούμενους λόγους προκύπτει ότι θα πρέπει πάντοτε ο αριθμός των κόμβων σε μια συστάδα εξυπηρετητών Γαλέρα να είναι περιττός ώστε να επιτευθεί η συγκατάθεση της πλειοψηφίας.

Σε σχέση με το Θεώρημα CAP η συστάδα Γαλέρα θεωρείτε ότι ικανοποιεί δύο από τις τρεις ιδιότητες, την ιδιότητα της συνέπειας καθώς και της ανοχής στις κατατμήσεις που οφείλονται στο δίκτυο. Αυτό συμβαίνει διότι για κάθε αλλαγή που πραγματοποιείτε σε μία βάση, τα δεδομένα συγχρονίζονται σε όλη την συστάδα και μόνο σε περίπτωση επιτυχής εγγραφής σε όλους τους κόμβους λαμβάνει κάποιος απάντηση. Επίσης διατηρείται η συνέπεια λόγω της ανάγκης συγκατάθεσης της πλειοψηφίας σε σχέση με το περιεχόμενο που διαθέτουν οι κόμβοι. Αντίθετα οι κόμβοι που ανήκουν στην μειοψηφία θα αρνηθούν κάθε διαδικασία εγγραφής και ανάγνωσης για να διατηρήσουν την ιδιότητα της συνέπειας, καθιστώντας τους εαυτούς τους μη διαθέσιμους.

2.1.3 Σύνδρομο Διάσπασης Εγκεφάλου (Split Brain Syndrome)

Ένα φαινόμενο που μπορεί να πραγματοποιηθεί σε μια συστάδα εξυπηρετητών είναι το σύνδρομο διάσπασης εγκεφάλου [18]. Στην περίπτωση μας είναι μια κατάσταση στην οποία μία συστάδα εξυπηρετητών διαχωρίζεται σε μικρότερες συστάδες ίσων κόμβων και κάθε μία πιστεύει ότι είναι η μοναδική ενεργή.

Πιστεύοντας ότι κάθε μία είναι μη ενεργή έχει σαν αποτέλεσμα την πιθανότητα κάθε μία να επιθυμεί να αποκτήσει πρόσβαση στα ίδια δεδομένα , το οποίο μπορεί να προκαλέσει αλλοίωση των δεδομένων (data corruption). Μια κατάσταση διάσπασης εγκεφάλου συμβαίνει κατά την διάρκεια της αναδιαμόρφωσης μιας συστάδας εξυπηρετητών. Όταν ένας ή περισσότεροι κόμβοι αποτυγχάνουν , τότε η συστάδα εξυπηρετητών αναδιαμορφώνεται με τους διαθέσιμους κόμβους. Κατά την διάρκεια της αναδιαμόρφωσης, αντί να δημιουργείται μια μοναδική συστάδα εξυπηρετητών , μπορεί να δημιουργηθούν πολλαπλά τμήματα της συστάδας με ίσο αριθμό κόμβων. Κάθε μια συστάδα πιστεύει ότι είναι η μοναδική ενεργή και ότι οι υπόλοιποι είναι μη ενεργοί και ως αποτέλεσμα ξεκινά να αποκτά πρόσβαση στα δεδομένα .Επειδή παραπάνω από μια συστάδα αποκτά πρόσβαση στα δεδομένα τα δεδομένα αλλοιώνονται.

Για την αποφυγή κάποιου τέτοιου φαινομένου , η συστάδα εξυπηρετητών Γαλέρα σε περίπτωση που δεν έχει επιτευθεί η πλειοψηφία των κόμβων παραμένει μη λειτουργική, μέχρι επιπλέον στοιχεία να μπορούν να παρασχεθούν από μια τρίτη οντότητα όπως ο χειριστής της συστάδας. Ο τελευταίος καλείται να επιλέξει ποιος κόμβος διαθέτει το πιο πρόσφατα περιεχόμενο, βρίσκοντας αυτόν που διαθέτει το πιο πρόσφατο αναγνωριστικό συναλλαγής και να εκκινήσει την συστάδα εξυπηρετητών από τον προηγούμενο. [19]

2.1.4 Πλεονεκτήματα συστάδας εξυπηρετητών Γαλέρα

Τα χαρακτηριστικά της συστάδας εξυπηρετητών Γαλέρα οδηγούν σε ένα σύνολο πλεονεκτημάτων όσο αφορά τις λύσεις ΣΔΒΔ (DBMS) για συστάδες εξυπηρετητών συμπεριλαμβάνοντας: [15]

- Μη καθυστέρηση (lag) σκλάβου.
- Μη απώλεια συναλλαγών.
- Κλιμάκωση εγγραφών και αναγνώσεων.
- Μικρότερες καθυστερήσεις πελατών.

2.2 Υψηλής Διαθεσιμότητας Εξυπηρετητής (HAProxy): Εισαγωγή

Οι εφαρμογές συνήθως συνδέονται σε μία συστάδα εξυπηρετητών μέσω του να ανοίγουν συνδέσεις με τους κόμβους της βάσης για να εκτελέσουν συναλλαγές. Σε περίπτωση αποτυχίας του κόμβου, ο πελάτης (client) θα πρέπει να ξανασυνδεθεί σε έναν άλλον κόμβο της βάσης πριν μπορέσει να συνεχίσει να εξυπηρετεί τις αιτήσεις του. [20]

Υπάρχουν διαφορετικοί τρόποι για να παρέχουμε συνδεσιμότητα σε μία ή περισσότερες συστάδες εξυπηρετητών MySQL. Ένας τρόπος είναι να χρησιμοποιήσουμε έναν οδηγό βάσης (database driver) ο οποίος υποστηρίζει ανακυκλώσιμες συνδέσεις (connection pooling), κατανομή φορτίου και διαδικασίες εναλλακτικής σύνδεσης (failover) για παράδειγμα:

- JDBC driver for MySQL (MySQL Connector/J)
- PHP MySQL native driver for master slave (mysqlnd-ms)

Οι προηγούμενοι οδηγοί βάσεων είναι φτιαγμένοι για να προσφέρουν διαφάνεια σε πελάτες όταν συνδέονται σε αυτόνομους (standalone) εξυπηρετητές MySQL, συστάδες

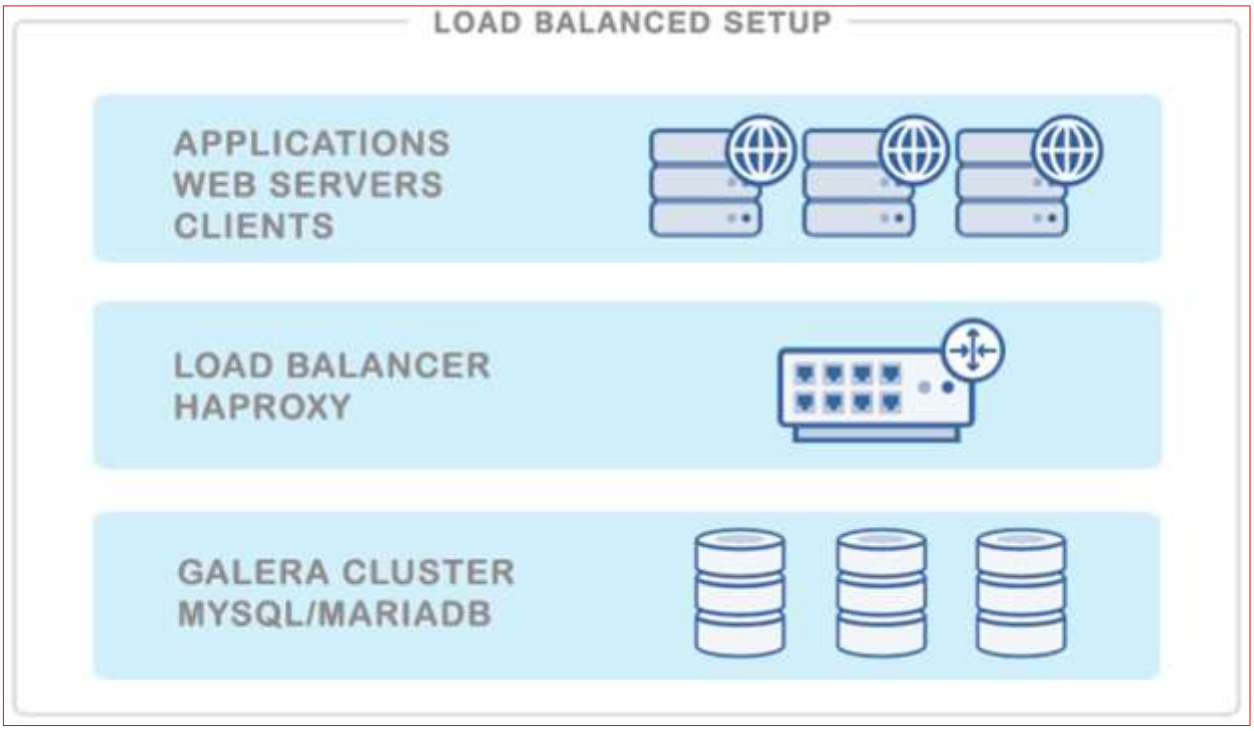
εξυπηρετητών MySQL ή εγκαταστάσεις αντιγραφής δεδομένων MySQL. Ωστόσο , σε άλλες εγκαταστάσεις συστάδων εξυπηρετητών, όπως η συστάδα εξυπηρετητών Γαλέρα για MySQL ή MariaDB, οι JDBC και PHP οδηγοί δεν είναι ενήμεροι για τις εσωτερικές πληροφορίες κατάστασης της Γαλέρα. Για παράδειγμα ένας Γαλέρα κόμβος μπορεί να βρίσκεται σε διαδικασία μόνο ανάγνωσης (read-only) ενώ βοηθάει έναν άλλο κόμβο να επανασυγχρονιστεί (resynchronize) (Αν η SST μέθοδος είναι mysqldump ή rsync) ή μπορεί να είναι σε μη πρωτεύουσα κατάσταση αν συμβεί φαινόμενο διάσπασης εγκεφάλου. Μια άλλη λύση είναι το να χρησιμοποιήσουμε έναν κατανεμητή φορτίου μεταξύ των πελατών και της συστάδας.

2.2.1 Λειτουργία HAProxy

Ο HAProxy είναι ένας υψηλής διαθεσιμότητας εξυπηρετητής μεσολάβησης και λειτουργεί ως TCP/HTTP εξισορροπιστής φορτίου (Load balancer). Κατανέμει το φόρτο εργασίας (workload) μεταξύ ενός συνόλου από εξυπηρετητές με σκοπό να μεγιστοποιήσει την απόδοση και την χρήση των πόρων.

Ο HAProxy διαθέτει μεθόδους (ελέγχου υγείας) health checks, επιτρέποντας ένα σύνολο από υπηρεσίες να μπορούν να εξισορροπήσουν από ένα μόνο τρέχων στιγμιότυπο.

Μία Front-end εφαρμογή (application) η οποία βασίζεται μία backend βάση μπορεί εύκολα να κατακλύσει την βάση με πάρα πολλές τρέχουσες συνδέσεις. Ο HAProxy παρέχει σειροποιησιμότητα και επιτάχυνση των συνδέσεων προς ένα ή περισσοτέρων εξυπηρετητών MySQL ενώ αποτρέπει ένα και μόνο εξυπηρετητή να κατακλυστεί από πάρα πολλά αιτήματα. Όλοι οι πελάτες συνδέονται στο στιγμιότυπο του HAProxy και ο αντίστροφος μεσολαβητής (reverse proxy) προωθεί την σύνδεση σε έναν από τους διαθέσιμους MySQL εξυπηρετητές σύμφωνα με τον αλγόριθμο εξισορρόπησης φορτίου που χρησιμοποιείται.



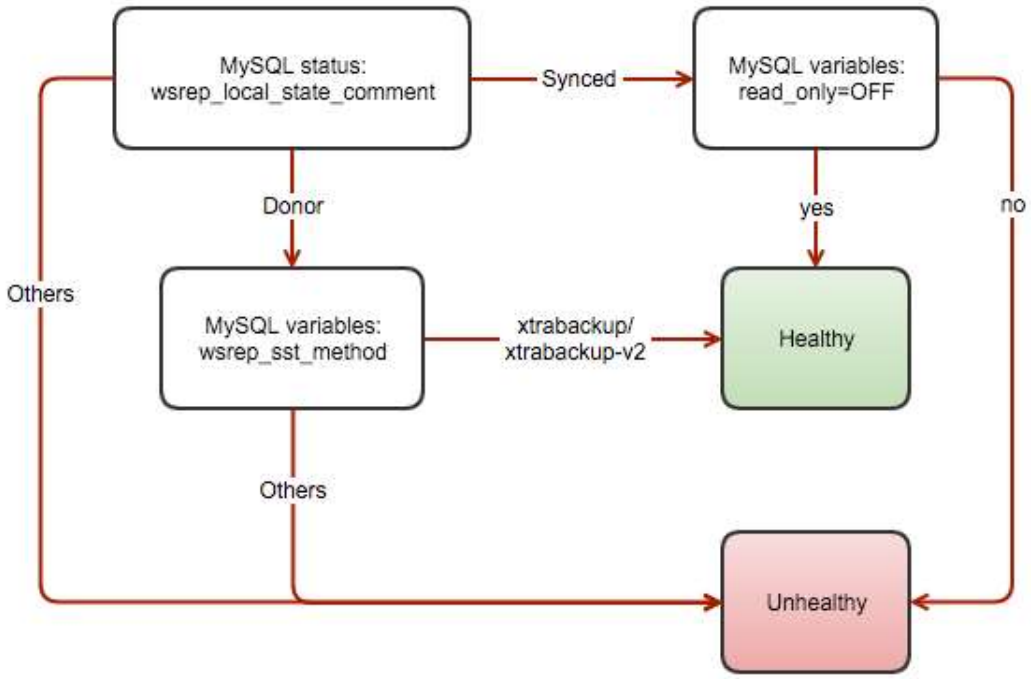
Εικόνα 8: Διάταξη εξισορρόπησης φορτίου.

Μια δυνατή εγκατάσταση θα ήταν να εγκατασταθεί ένας HAProxy σε κάθε έναν εξυπηρετητή δικτύου ή εξυπηρετητή εφαρμογής (web server ή application server). Αυτό θα λειτουργούσε με καλά αποτελέσματα αν υπήρχε ένας μικρός αριθμός από

εξυπηρετητές δικτύου, ώστε το φορτίο το οποίο θα παρεχόταν από τους ελέγχους υγείας να ελέγχεται. Ο εξυπηρετητής δικτύου θα συνδεόταν με τον τοπικό HAProxy (πχ κάνοντας μια mysql σύνδεση στο 127.0.0.1:3306) και θα μπορούσε να έχει πρόσβαση σε όλους τους εξυπηρετητές βάσεων δεδομένων. Ο Εξυπηρετητής δικτύου και ο HAProxy μαζί αποτελούν μια λειτουργική οντότητα, έτσι ώστε ο εξυπηρετητής δικτύου δεν θα λειτουργεί αν ο HAProxy δεν είναι διαθέσιμος.

Με τον HAProxy σαν εξισορροπιστή φορτίου θα έχουμε τα παρακάτω πλεονεκτήματα:

- Όλες οι εφαρμογές έχουν πρόσβαση στην συστάδα μέσω μίας και μόνο διεύθυνσης IP (IP address) ή όνομα διακομιστή (hostname). Με αποτέλεσμα η τοπολογία της συστάδας να είναι κρυμμένη πίσω από τον HAProxy.
- Οι MySQL συνδέσεις είναι εξισορροπημένες με βάση το φορτίο μεταξύ των διαθέσιμων κόμβων της βάσης.
- Είναι δυνατόν να αφαιρέσουμε ή να προσθέσουμε κόμβους χωρίς καμία αλλαγή στην εφαρμογή μας.
- Όταν φτάσουμε τον μέγιστο δυνατό αριθμό συνδέσεων στην βάση μας , ο HAProxy θέτει σε ουρά προτεραιότητας τις επιπλέον συνδέσεις. Με αυτό τον τρόπο επιτυγχάνει να ελέγχει τις συνδέσεις προς την βάση και να αποφεύγει την κατάκλυση από αιτήσεις.



Εικόνα 9: Παράδειγμα Αρχιτεκτονικής HAProxy

2.2.2 Έλεγχοι υγείας για MySQL.

Είναι δυνατό να ορίσουμε τον HAProxy να ελέγχει ότι ο ένας εξυπηρετητής είναι ενεργώς, με το να κάνει μια σύνδεση με την θύρα MySQL (συνήθως 3306). Παρόλα αυτά, αυτό δεν είναι αρκετό. Το στιγμιότυπο θα μπορούσε να είναι ενεργό, αλλά ο μηχανισμός αποθήκευσης δεδομένων μπορεί να μην λειτουργεί όπως θα έπρεπε. Υπάρχουν συγκεκριμένοι έλεγχοι .που πρέπει να γίνουν, εξαρτώμενοι αν η συστάδα είναι Γαλέρα, Mysql Αντιγραφή δεδομένων ή Συστάδα Εξυπηρετητών MySQL.

Ο HAProxy διαπιστώνει αν ένας εξυπηρετητής είναι διαθέσιμος για αίτηση δρομολόγησης εφαρμόζοντας ελέγχους υγείας. Στην διάταξή μας θα χρησιμοποιούμε

ένα HTTP πρωτόκολλο (httpchk) για να ελέγχουμε την υγεία των εξυπηρετητών μας. Αυτό είναι κάτι συνηθισμένο αν επιθυμούμε να πραγματοποιούμε εξισορρόπηση φορτίου σε μια υπηρεσία HTTP, όπου ο HAProxy επιτυγχάνει ότι το backend επιστρέφει μια συγκεκριμένη HTTP απάντηση πριν δρομολογηθούν οι εισερχόμενες συνδέσεις. Αυτή η μέθοδος είναι προτιμητέα καθότι χρησιμοποιεί λιγότερους πόρους με μια σύνδεση HTTP χωρίς να τηρεί κατάσταση (stateless HTTP connection).

Η καλύτερη πρακτική για να εφαρμόσουμε ελέγχους υγείας MySQL είναι με το να χρησιμοποιήσουμε ένα αρχείο κώδικα το οποίο θα καθορίζει πότε ένας εξυπηρετητής MySQL είναι διαθέσιμος με το να ελέγχει προσεκτικά την εσωτερική του κατάσταση, η οποία εξαρτάται από την διαμόρφωση της συστάδας στην οποία ανήκει.

Ένα τέτοιο παράδειγμα είναι το αρχείο κώδικα που δημιουργήσαμε και ονομάσαμε clustercheck (/usr/bin/clustercheck). Αυτό το αρχείο βρίσκεται σε κάθε MySQL εξυπηρετητή στην διαμόρφωση εξισορρόπησης φορτίου και έχει την δυνατότητα να επιστρέψει μια HTTP απάντηση σαν καθορισμένη έξοδο που είναι χρήσιμο στους ελέγχους υγείας.

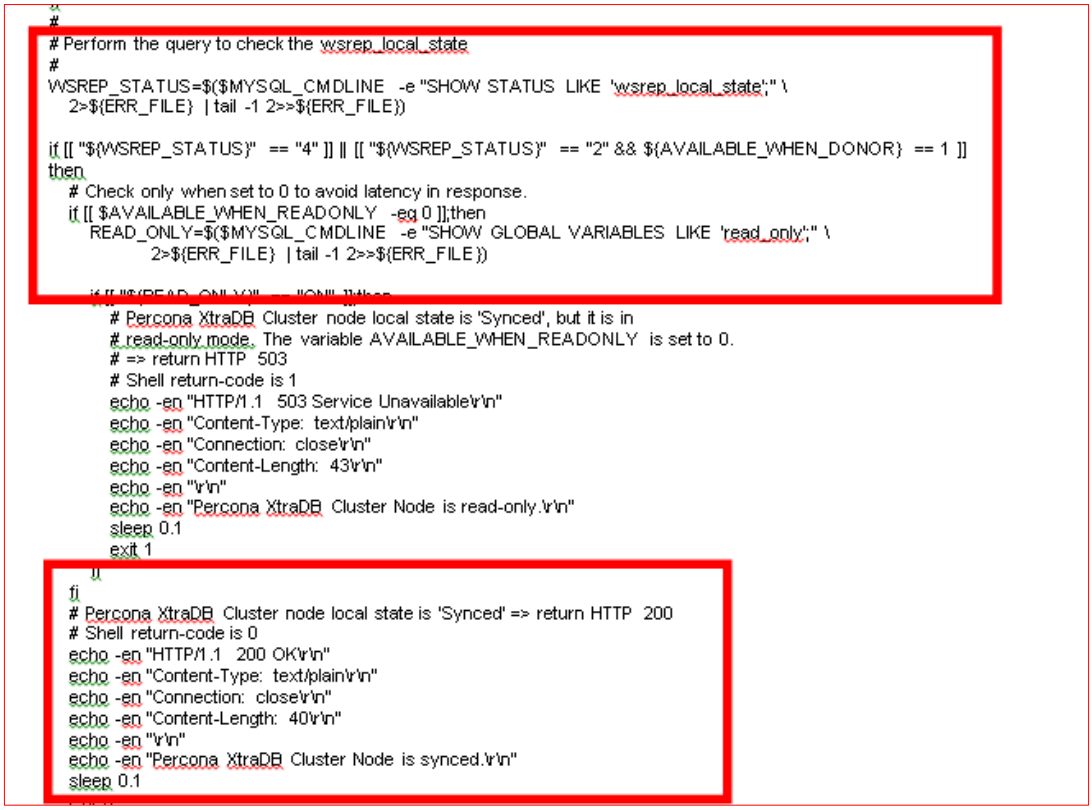
Αν ο backend MySQL εξυπηρετητής είναι υγιής , τότε το αρχείο κώδικα θα επιστρέψει ένα απλό “HTTP 200 OK status” με “exit status 0” . Αλλιώς θα επιστρέψει ένα “503 Service unavailable” και “exit status 1”.

Αυτό το αρχείο κώδικα θα εκτελείται από μια υπηρεσία, την mysqlchk του δαίμονα (daemon) xinetd (extended internet daemon). Ο δαίμονας θα έχει σαν αποστολή να ακούει μια συγκεκριμένη θύρα (σαν αρχική επιλογή είναι η 9200).

```
  GNU nano 2.3.1            File: /etc/xinetd.d/mysqlchk

# default: on
# description: mysqlchk
service mysqlchk
{
 flags = REUSE
 socket_type = stream
 port = 9200
 wait = no
 user = nobody
 server = /usr/bin/clustercheck
 log_on_failure += USERID
 disable = no
 only_from = 0.0.0.0/0
 per_source = UNLIMITED
}
```

Εικόνα 10: Αρχείο Διαμόρφωσης /etc/xinetd.d/mysqlchk

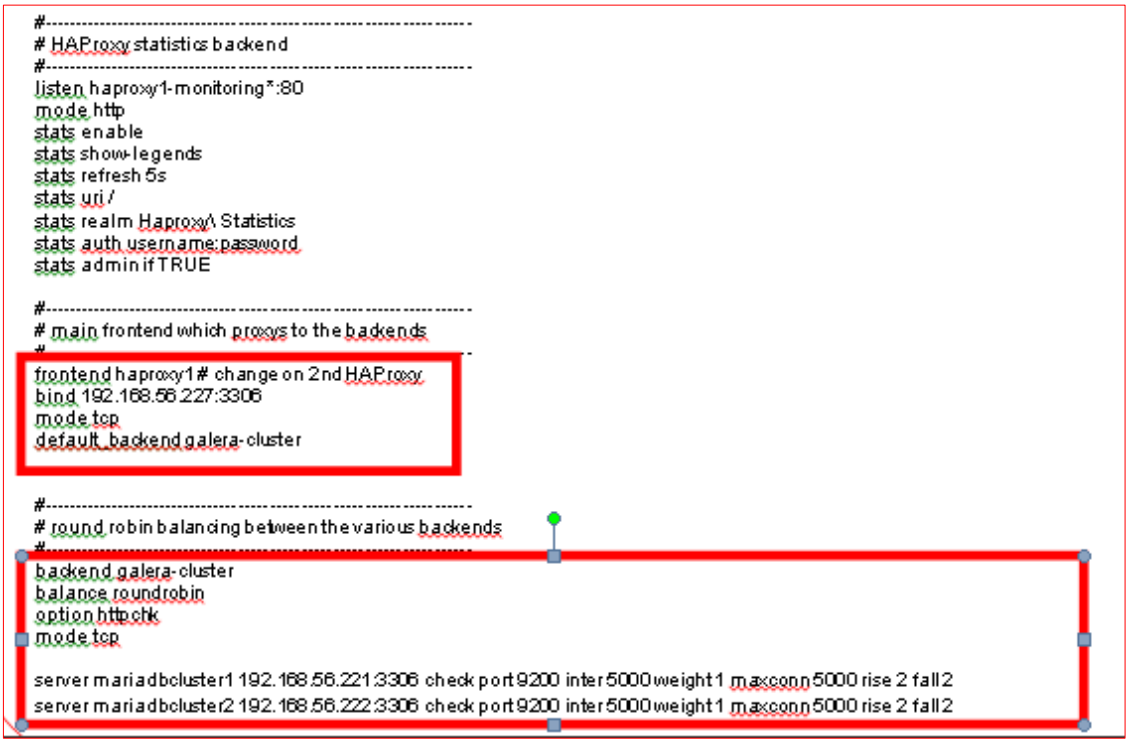
Ο HAProxy , τότε θα συνδεθεί σε αυτή την θύρα και θα περιμένει για την εξαγόμενη HTTP απάντηση, για να ελέγξει αν θα δρομολογήσει τα αίτημα προς αυτόν τον κόμβο σύμφωνα με τον αλγόριθμο εξισορρόπησης φορτίου που χρησιμοποιείται.



Εικόνα 11:Τμήμα του αρχείου κώδικα /usr/bin/clustercheck

2.2.3 Αναγνώριση μη διαθεσιμότητας του κόμβου.

Υπάρχουν πολλές παράμετροι καθορισμένες από το χρήστη που καθορίζουν πόσο γρήγορα θα είναι σε θέση ο HAProxy να εντοπίσει ότι ένας διακομιστής δεν είναι διαθέσιμος. Τις ρυθμίσεις για την προηγούμενη δραστηριότητα τις πραγματοποιήσαμε στο /etc/haproxy/haproxy.cfg.



Εικόνα 12: Τμήμα του αρχείου /etc/haproxy/haproxy.cfg.

Γρήγορη εξήγηση για κάθε γραμμή παραπάνω:

- Listen: Η ενότητα Listen ορίζει έναν πλήρη διακομιστή μεσολάβησης με τα τμήματα του frontend και του backend που συνδυάζονται σε ένα τμήμα. Είναι γενικά χρήσιμο για κυκλοφορία μόνο με TCP.
- bind: Συνδέστε όλες τις διευθύνσεις IP σε αυτόν τον κεντρικό υπολογιστή στη θύρα 3307. Οι πελάτες σας θα πρέπει να συνδεθούν στη θύρα που ορίζεται σε αυτή τη γραμμή.
- Mode: Πρωτόκολλο του στιγμιότυπου. Για την MySQL, το στιγμιότυπο θα πρέπει να λειτουργεί σε καθαρή λειτουργία TCP. Θα δημιουργηθεί μια πλήρη αμφίδρομη σύνδεση μεταξύ των υπολογιστών-πελατών και των διακομιστών και δεν θα πραγματοποιηθεί εξέταση του επιπέδου 7.
- balance: Ο αλγόριθμος εξισορρόπησης φορτίου. είναι σε θέση να υποστηρίξει τιμές όπως leastconn, roundrobin και source,
- leastconn – Ο εξυπηρετητής με τον λιγότερο αριθμό από συνδέσεις δέχεται την σύνδεση.
- roundrobin – Κάθε ένας εξυπηρετητής χρησιμοποιείτε περιοδικά, ανάλογα το βάρος που του έχουμε δώσει. (Η επιλογή που έχουμε κάνει στο πείραμά μας).
- source – Χρησιμοποιείται ο ίδιος εξυπηρετητής που έχουμε ορίσει.
- httpchk: Εκτελέστε τον έλεγχο υγείας βάσει HTTP. Αναφέρεται στο αρχείο κώδικα που θέσαμε στην θέση /usr/bin/clustercheck σε κάθε ένα backend εξυπηρετητή επιστρέφει τον κώδικα απόκρισης HTTP.
- Default-server: Προεπιλεγμένες επιλογές για τους backend εξυπηρετητές που αναφέρονται στην επιλογή server.
- port: Η θύρα ελέγχου υγείας backend. Ο xinetd που ακούει στη θύρα 9200 σε κάθε κόμβο βάσης δεδομένων , τρέχει ένα προσαρμοσμένο σενάριο ελέγχου υγείας.
- Inter: Το διάστημα μεταξύ ελέγχων υγείας για ένα διακομιστή που είναι "επάνω", μεταβατικά "πάνω ή κάτω" ή δεν έχει ακόμα ελεγχθεί είναι 5 δευτερόλεπτα.
- weight: Στη Γαλέρα, όλοι οι κόμβοι αντιμετωπίζονται συνήθως εξίσου.
- Maxconn: Ο HAProxy θα σταματήσει να δέχεται συνδέσεις όταν ο αριθμός συνδέσεων είναι 5000.
- rise: Ο εξυπηρετητής θα θεωρείται διαθέσιμος μετά από 3 διαδοχικούς επιτυχείς υγειονομικούς ελέγχους.
- fall: Ο διακομιστής θα θεωρείται μη διαθέσιμος μετά από 2 διαδοχικούς ανεπιτυχείς ελέγχους υγείας.

Από τις παραπάνω διαμορφώσεις, ο backend MySQL εξυπηρετητής αποτυγχάνει σε ελέγχους υγείας όταν:

- Ο HAProxy δεν μπόρεσε να συνδεθεί στη θύρα 9200 του εξυπηρετητή MySQL.
- Αν η θύρα 9200 είναι συνδεδεμένη, ο κώδικας απόκρισης HTTP που αποστέλλεται από το εξυπηρετητή MySQL επιστρέφει κάτι εκτός από το HTTP / 1.1 200 OK (επιλογή httpchk).

Ως εκ τούτου, η χρονική περίοδος διαθεσιμότητας και μη διαθεσιμότητας θα είναι:

- Κάθε 5 δευτερόλεπτα, ο HAProxy εκτελεί έλεγχο υγείας στη θύρα 9200 του εξυπηρετητή backend (θύρα 9200 inter 5000).
- Αν ο έλεγχος υγείας αποτύχει, ξεκινά η μέτρηση του χρόνου μη διαθεσιμότητας και θα ελέγξει για την επόμενη αποτυχία. Μετά από 5 δευτερόλεπτα, αν η δεύτερη προσπάθεια εξακολουθεί να αποτυγχάνει, ο HAProxy θα σημάνει το MySQL server ως μη διαθέσιμο (inter 5000 fall 2).

- Ο εξυπηρετητής MySQL αποκλείεται αυτόματα από τη λίστα των διαθέσιμων εξυπηρετητών.
- Μόλις ο εξυπηρετητής MySQL γίνει πάλι διαθέσιμος, αν ο έλεγχος υγείας πετύχει, ξεκινάει ένας αύξων αριθμός, ο οποίος θα ελέγξει εάν επιτυγχάνεται η επόμενη διαδοχική προσπάθεια. Αν ο αριθμός φτάσει στο 3, ο εξυπηρετητής θα επισημανθεί ως διαθέσιμος (rise 3).
- Ο εξυπηρετητής MySQL συμπεριλαμβάνεται αυτόματα στη λίστα των διαθέσιμων εξυπηρετητών.
- Ο εξυπηρετητής MySQL είναι σε πλήρη λειτουργία.

2.2.4 Διαχωρισμός ανάγνωσης / εγγραφής με HAProxy.

Ο HAProxy ως MySQL load balancer λειτουργεί παρόμοια με μια συσκευή που προωθεί αιτήματα TCP, το οποίο λειτουργεί στο επίπεδο μεταφοράς του μοντέλου TCP / IP. Δεν κατανοεί τα ερωτήματα MySQL (που λειτουργεί στο υψηλότερο επίπεδο) , που διανέμει στους backend εξυπηρετητές MySQL. Εξαιτίας αυτού, ο HAProxy είναι δημοφιλές μεταξύ εγκαταστάσεων πολλαπλών αφεντών όπως το Galera Cluster και το MySQL Cluster, όπου όλοι οι εξυπηρετητές MySQL υποστηρίζονται εξίσου. Όλοι οι εξυπηρετητές MySQL είναι σε θέση να χειριστούν τις προωθημένες διαδικασίες ανάγνωσης / εγγραφής. Το γεγονός ότι λειτουργεί στο επίπεδο μεταφοράς, οδηγεί στο να καταναλώνει επίσης λιγότερο φορτίο, σε σύγκριση με έναν εξισορροπιστή φορτίου / αντίστροφης μεσολαβητής όπως το MaxScale ή ProxySQL.

2.3 Διαθεσιμότητα του HAProxy μέσω του Keepalived.

Αρχικά θα ορίσουμε τον εικονικό εξυπηρετητή IP (IP Virtual Server - IPVS)

Ο IPVS υλοποιεί την εξισορρόπηση φορτίου στο επίπεδο μεταφοράς μέσα στον πυρήνα του Linux και ονομάζεται εναλλάκτης επιπέδου - 4. Το IPVS που εκτελείται σε ένα κεντρικό υπολογιστή λειτουργεί ως εξισορροπιστής φορτίου στο μπροστινό μέρος μια συστάδας πραγματικών εξυπηρετητών, μπορεί να κατευθύνει τις αιτήσεις για υπηρεσίες βασισμένες σε TCP / UDP στους πραγματικούς εξυπηρετητές και να κάνει τις υπηρεσίες των πραγματικών εξυπηρετητών να εμφανίζονται ως εικονική υπηρεσία σε μια ενιαία διεύθυνση IP. [21]

Το IPVS παρακολουθεί την κατάσταση κάθε εξυπηρετητή και χρησιμοποιεί το πρωτόκολλο εφεδρείας εικονικού δρομολογητή (Virtual Router Redundancy Protocol - VRRP) για την εφαρμογή υψηλής διαθεσιμότητας. Το Keepalived χρησιμοποιεί το IPVS για την παροχή εξισορρόπηση φορτίου στο επίπεδο μεταφοράς (Layer 4) και την ανακατεύθυνση αιτημάτων για υπηρεσίες που βασίζονται στο δίκτυο σε μεμονωμένα μέλη μιας συστάδας εξυπηρετητών. [22]

Μια τυπική διαμόρφωση υψηλής διαθεσιμότητας αποτελείται από ένα κύριο εξυπηρετητή και έναν ή περισσότερους διακομιστές αντιγράφων ασφαλείας. Μία ή περισσότερες εικονικές διευθύνσεις IP, που ορίζονται ως στιγμιότυπα VRRP, εκχωρούνται στις διεπαφές δικτύου του κεντρικού διακομιστή έτσι ώστε να μπορούν να εξυπηρετούν πελάτες δικτύου. Οι διακομιστές αντιγράφων ασφαλείας αναμένουν να ακούν πακέτα διαφήμισης VRRP πολλαπλής διανομής τα οποία ο κύριος διακομιστής εκπέμπει σε τακτά χρονικά διαστήματα. Το προεπιλεγμένο διάστημα διαφήμισης είναι ένα δευτερόλεπτο. Εάν οι κόμβοι αντιγράφων ασφαλείας δεν λάβουν τρεις διαδοχικές διαφημίσεις VRRP, ο διακομιστής δημιουργίας αντιγράφων ασφαλείας με την υψηλότερη προτεραιότητα αναλαμβάνει ως κύριος εξυπηρετητής και εκχωρεί τις εικονικές διευθύνσεις IP στις δικές του διεπαφές δικτύου. Εάν αρκετοί διακομιστές δημιουργίας αντιγράφων ασφαλείας έχουν την ίδια προτεραιότητα, ο διακομιστής

αντιγράφων ασφαλείας με την υψηλότερη τιμή διεύθυνσης IP γίνεται ο κύριος διακομιστής.

Το αρχείο διαμόρφωσης, για τον διαχειριζόμενο δαίμονα, είναι το /etc/keepalived/keepalived.conf. Αυτό το αρχείο πρέπει να υπάρχει σε κάθε διακομιστή στον οποίο έχετε ρυθμίσει το Keepalived για εξισορρόπηση φορτίου ή υψηλή διαθεσιμότητα

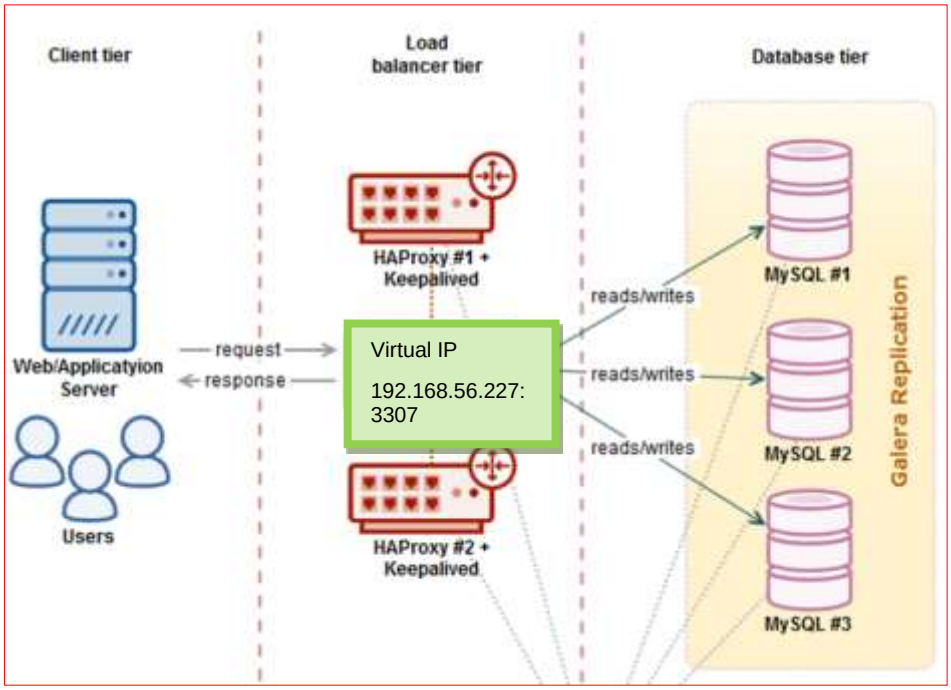
Στο επόμενο στιγμιότυπο ο HAProxy1 έχει λάβει προτεραιότητα 101 ενώ ο HAProxy2 έχει λάβει προτεραιότητα 100.

```
  GNU nano 2.3.1        File: /etc/keepalived/keepalived.conf

! Configuration File for keepalived

global_defs {
router_id haproxy1 # hostname

notification_email {
msen@na.edu
}
notification_email_from haproxy1@na.edu
smtp_server 127.0.0.1
smtp_connect_timeout 30
}

vrrp_script check_haproxy {
script "killall -0 haproxy"
interval 2 # every 2 seconds
weight 2 # add 2 points if OK
}

vrrp_instance VI_1 {
state MASTER # MASTER on haproxy1, BACKUP on haproxy2
interface enp0s8 # interface to monitor
virtual_router_id 51
priority 101 # 101 on haproxy1, 100 on haproxy2
advert_int 1

virtual_ipaddress {
192.168.56.227  dev enp0s8 #virtual ip address
}
track_script {
check_haproxy
}
}
```

Εικόνα 13: Αρχείο Διαμόρφωσης /etc/keepalived/keepalived.conf

Δεδομένου ότι όλες οι εφαρμογές θα εξαρτώνται από τον HAProxy για να συνδεθούνε με έναν διαθέσιμο κόμβο βάσης δεδομένων, για να αποφύγουμε ένα μόνο σημείο αποτυχίας με το HAProxy μας, θα δημιουργούσατε δύο πανομοιότυπες παρουσίες HAProxy (μία ενεργή και μία σε αναμονή) και θα χρησιμοποιήσουμε το Keepalived για να εκτελέσουμε VRRP μεταξύ τους . Το VRRP παρέχει μια εικονική διεύθυνση IP στο ενεργό HAProxy και μεταφέρει την εικονική διεύθυνση IP στο κατάσταση αναμονής HAProxy σε περίπτωση μη διαθεσιμότητας του πρώτου. Αυτό είναι απρόσκοπτο επειδή οι δύο περιπτώσεις HAProxy δεν χρειάζονται κοινή κατάσταση.

Με την προσθήκη του Keepalived στην εικόνα, η αρχιτεκτονική μας θα μοιάζει με κάτι τέτοιο:



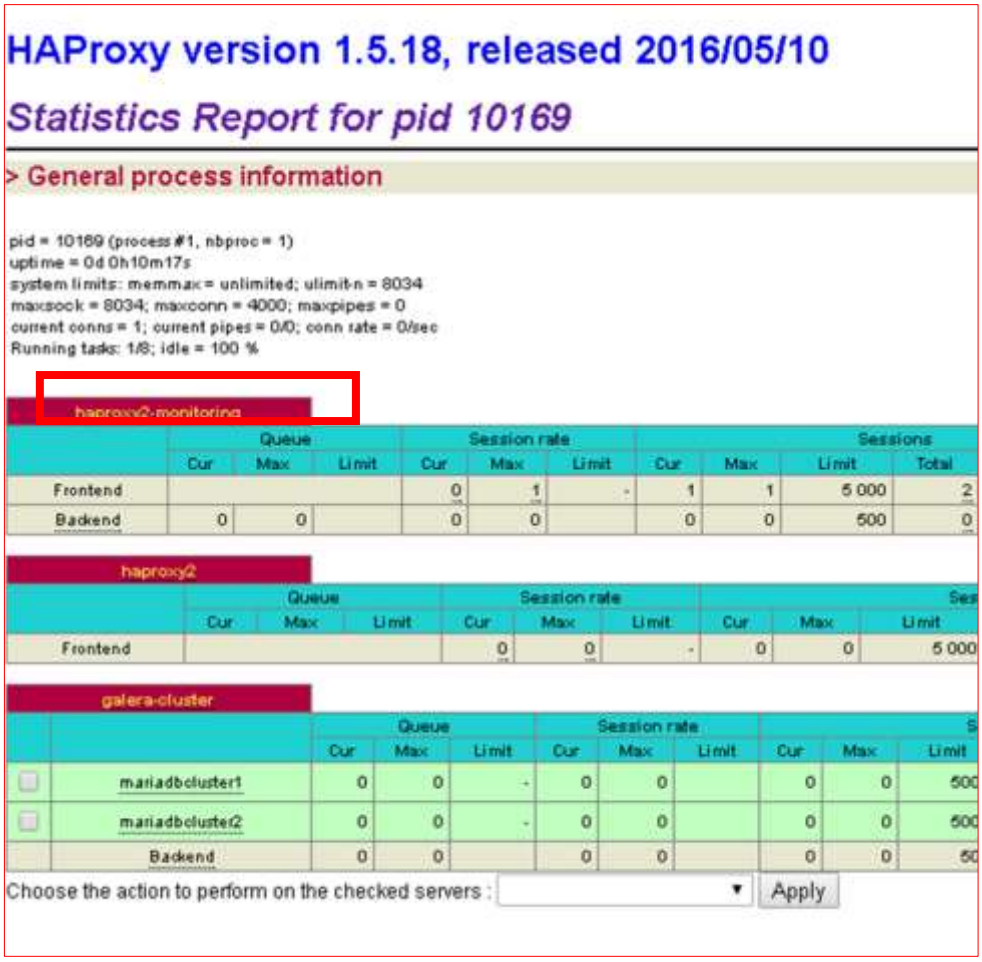
Εικόνα 14: Προσθήκη keepalived στην αρχιτεκτονική μας.

Σε αυτό το παράδειγμα, χρησιμοποιούμε δύο κόμβους για να λειτουργήσουνε ως εξισορροπιστής φορτίου με ανακατεύθυνση διεύθυνσης IP μπροστά από την συστάδα των βάσεων δεδομένων μας. Η εικονική διεύθυνση IP (Virtual IP - VIP) θα κυμαίνεται μεταξύ HAProxy # 1 (αφέντη) και HAProxy # 2 (αναμονή). Όταν ο HAProxy # 1 τεθεί μη διαθέσιμος, ο HAProxy # 2 θα αναλάβει την VIP και μόλις επανέλθει σε διαθεσιμότητα ο HAProxy # 1, το VIP θα ανατεθεί στο HAProxy # 1 αφού κατέχει τον υψηλότερο αριθμό προτεραιότητας. Η διαδικασία ανάθεσης είναι αυτόματη, ελέγχεται από το Keepalived. Απαραίτητη είναι η ύπαρξη δύο εξυπηρετητών τουλάχιστον για να πραγματοποιηθεί η προηγούμενη διαδικασία.

2.3.1 Αναφορά Στατιστικών.

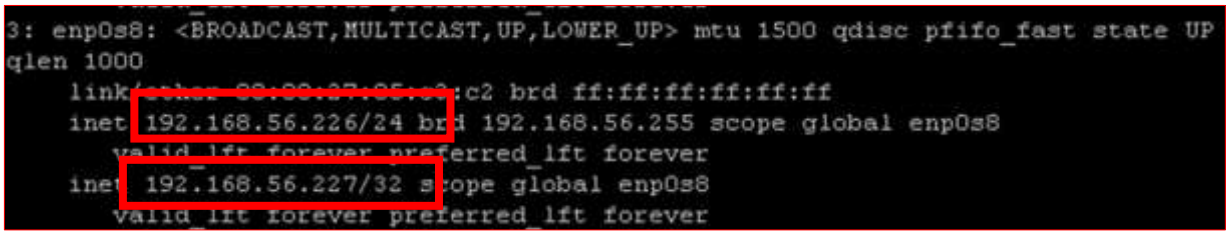
Είναι επίσης δυνατή η πρόσβαση στην προεπιλεγμένη σελίδα στατιστικών του HAProxy συνδέοντας τη θύρα 80 στον κόμβο HAProxy. Ένα παράδειγμα για το προηγούμενο θα ήταν: http: // [HAProxy_node_IP_address]: 80 / με όνομα χρήστη username και κωδικό πρόσβασης αυτά που θέσαμε στο /etc/haproxy/haproxy.cfg.

Το ακόλουθο στιγμιότυπο οθόνης δείχνει το παράδειγμα κατά τη σύνδεση στην IP http://192.168.56.227/ με όνομα χρήστη “username” και κωδικό πρόσβασης "password":



Εικόνα 15: Σελίδα στατιστικών HAProxy2

Στο προηγούμενο στιγμιότυπο παρατηρούμε ότι χρησιμοποιήσαμε την VIP της αρχιτεκτονικής μας η οποία και έχει ανατεθεί στον HAProxy2 και σαν αποτέλεσμα έχουμε πρόσβαση στην σελίδα στατιστικών του. Ο HAProxy2 διαθέτει την VIP, ενώ ο HAProxy1 δεν παρουσιάζεται, περιμένει πότε ο HAProxy1 θα καταστεί μη διαθέσιμος για να αναλάβει την VIP. Σαν αποτέλεσμα ο HAProxy2 ανακατευθύνει τα αιτήματα στην συστάδα εξυπηρετητών που αποτελούν την βάση δεδομένων μας.



Εικόνα 16: Ανάθεση VIP στον HAProxy2

2.3.2 Πλεονεκτήματα Keepalived- HAProxy.

Το Keepalived είναι επιπέδου πυρήνα και δρομολογεί πακέτα, σαν ένας δρομολογητής.

Από τα παραπάνω προκύπτουν τα παρακάτω συμπεράσματα:

- Αν επιθυμούμε εξισορρόπηση φορτίου βασιζόμενη απόλυτα σε αριθμό συνδέσεων ή μας ενδιαφέρει το φορτίο που δέχεται η ΚΜΕ, τότε το Keepalived όντας επιπέδου 4 εξισορροπιστής φορτίου είναι αυτό που χρειαζόμαστε. Μας παρέχει υψηλή ταχύτητα, μικρή καταπόνηση της ΚΜΕ και υψηλό αριθμό συναλλαγών το δευτερόλεπτο .
- Αν επιθυμούμε ο εξισορροπιστής φορτίου να αντιλαμβάνεται τα πρωτόκολλα δρομολόγησης των backend εξυπηρετητών, να κάνει έλεγχο των πακέτων , τότε αναφερόμαστε σε εξισορροπιστή φορτίου επιπέδου 7 και θα πρέπει να χρησιμοποιήσουμε τους HAProxy.

Συνοψίζοντας τα χαρακτηριστικά που έχουμε διαπιστώσει αυτή την στιγμή στην αρχιτεκτονική μας είναι:

Υψηλή διαθεσιμότητα.

- Καθώς το keepalived θα διαλέγει πάντα ένα κύριο κόμβο από τους δύο HAProxy, που θα του παρέχει αυτή την διεύθυνση.
- Κάθε keepalived στιγμιότυπο , θα κάνει έλεγχο καλής κατάστασης για τον HAProxy
- Παρέχει με αυτοματοποιημένη διαδικασία με τρόπο διαφανή στον χρήση σε περίπτωση μη διαθεσιμότητας του κύριου κόμβου, την VIP σε κάποιον άλλο κόμβο.

Εξισορρόπηση φορτίου:

- Κάθε HAProxy στιγμιότυπο είναι συνδεδεμένη με ένα MariaDB στιγμιότυπο, παρέχοντας τους εξισορρόπηση φορτίου.
- Επιπλέων έλεγχοι γίνονται για την κατάσταση των στιγμιότυπων της MariaDB μέσω του δαίμονα xinetd και του αρχείου κώδικα clustercheck που δημιουργήσαμε.

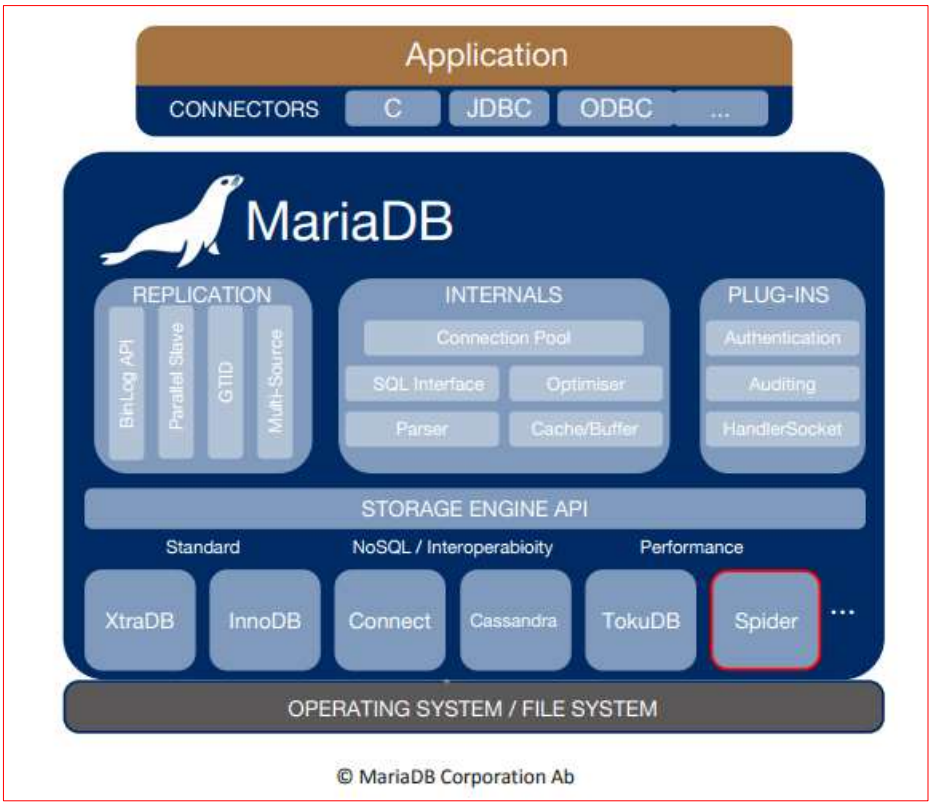
Αντιγραφή Δεδομένων

- Η αντιγραφή δεδομένων είναι το τελευταίο επιθυμητό χαρακτηριστικό της διάταξης των πολλαπλών αφεντών της συστάδας εξυπηρετών Γαλέρα.

2.4 Μηχανισμός Αποθήκευσης Δεδομένων Spider: Εισαγωγή

Όταν τα μεγέθη αποθήκευσης δεδομένων μας αρχίζουν να φτάνουν τα όρια των μεγάλων δεδομένων, ο κατακερματισμός δεδομένων γίνεται απαραίτητος. Ωστόσο, μπορεί να χρειαστεί να χρησιμοποιηθεί και για άλλους λόγους, όπως η αντιμετώπιση τεράστιων ποσοτήτων κυκλοφορίας δεδομένων. Υπάρχουν πολλοί τρόποι για να πραγματοποιηθεί ο κατακερματισμός και αυτό έχει μεγάλο αντίκτυπο στην ίδια την απόδοση. Ένας αποδοτικός τρόπος για να πραγματοποιηθεί το προηγούμενο είναι η μηχανή αποθήκευσης δεδομένων Spider. [23]

Ο μηχανισμός αποθήκευσης Spider διαθέτει ενσωματωμένα χαρακτηριστικά κατακερματισμού. Υποστηρίζει κατατμήσεις και συναλλαγές XA (XA transactions) και επιτρέπει στους πίνακες διαφορετικών στιγμιότυπων της MariaDB να αντιμετωπίζονται σαν να ήταν το ίδιο στιγμιότυπο. [24]



Εικόνα 17: Αρχιτεκτονική μηχανισμών MariaDB

Οι συναλλαγές XA [25] έχουν σχεδιαστεί για να επιτρέπουν κατανεμημένες συναλλαγές, όπου ένας διαχειριστής συναλλαγών (η εφαρμογή) ελέγχει μια συναλλαγή που περιλαμβάνει πολλαπλούς πόρους. Οι πόροι αυτοί είναι συνήθως ένα ΣΔΒΔ, αλλά θα μπορούσαν να είναι πόροι οποιουδήποτε τύπου. Το σύνολο των απαιτούμενων πράξεων συναλλαγής ονομάζεται συνολική συναλλαγή. Κάθε υποσύνολο λειτουργιών που περιλαμβάνει έναν μόνο πόρο ονομάζεται τοπική συναλλαγή. Το XA χρησιμοποιεί μια δέσμευση 2 φάσεων (2PC). Με την πρώτη δέσμευση, ο διαχειριστής συναλλαγών λέει σε κάθε πόρο να προετοιμάσει μια αποτελεσματική δέσμευση και περιμένει ένα μήνυμα επιβεβαίωσης. Οι αλλαγές δεν εφαρμόζονται ακόμη σε αυτό το σημείο. Εάν οποιοσδήποτε από τους πόρους αντιμετώπισε ένα σφάλμα, ο διαχειριστής συναλλαγών θα ανακτήσει την παγκόσμια συναλλαγή. Εάν όλοι οι πόροι επικοινωνούν ότι η πρώτη δέσμευση είναι επιτυχής, ο χειριστής συναλλαγών μπορεί να απαιτήσει μια δεύτερη δέσμευση, πράγμα που καθιστά τις αλλαγές αποτελεσματικές

Στη MariaDB, οι συναλλαγές XA μπορούν να χρησιμοποιηθούν μόνο με μηχανισμούς αποθήκευσης που τις υποστηρίζουν. Τουλάχιστον οι InnoDB, TokuDB και Spider τις υποστηρίζουν

Όταν δημιουργείται λοιπόν ένας πίνακας με τον μηχανισμό αποθήκευσης Spider, ο προηγούμενος πίνακας συνδέεται με έναν πίνακα σε έναν απομακρυσμένο εξυπηρετητή. Ο απομακρυσμένος πίνακας μπορεί να έχει οποιαδήποτε μηχανισμό αποθήκευσης. Η διαδικασίας σύνδεσης επιτυγχάνεται μέσω της σύνδεσης του τοπικού εξυπηρετητή MariaDB με τον απομακρυσμένο.

Η εγκατάσταση του Spider μπορεί να γίνει μέσα από την mysql καθώς είναι διαθέσιμη από την έκδοση MariaDB 10.0.4.

```
MariaDB [(none)]> source /usr/share/mysql/install_spider.sql
Query OK, 0 rows affected (0.00 sec)
```

Εικόνα 18: Εγκατάσταση μηχανισμού αποθήκευσης δεδομένων Spider.

Αποτέλεσμα του προηγούμενου θα είναι να εμφανιστεί ο μηχανισμός Spider ανάμεσα στους υποστηριζόμενους μηχανισμούς αποθήκευσης δεδομένων.

```
MariaDB [(none)]> SELECT engine, support, transactions, xa FROM information_s
ma.engines;
+--------------------+---------+--------------+------+
| engine             | support | transactions | xa   |
+--------------------+---------+--------------+------+
| SPIDER             | YES     | YES          | YES  |
| CSV                | YES     | NO           | NO   |
| MRG_MyISAM         | YES     | NO           | NO   |
| BLACKHOLE          | YES     | NO           | NO   |
```

Εικόνα 19: Επιτυχής Εγκατάσταση μηχανισμού αποθήκευσης δεδομένων Spider.

Η προηγούμενη εγκατάσταση οδηγεί στην δημιουργία των επόμενων πινάκων στην βάση δεδομένων mysql:

- spider_link_failed_log
- spider_link_mon_servers
- spider_tables
- spider_xa
- spider_xa_failed_log
- spider_xa_member

2.4.1 Λειτουργία spider.

Μια τυπική ανάπτυξη Spider έχει μια αρχιτεκτονική χωρίς να διαμοιράζεται τίποτα μεταξύ των συστάδων εξυπηρετητών. Το σύστημα λειτουργεί με οποιοδήποτε φθηνό υλισμικό και με ελάχιστες ειδικές απαιτήσεις για υλισμικό ή λογισμικό. Αποτελείται από ένα σύνολο υπολογιστών, με μία ή περισσότερες διεργασίες MariaDB γνωστές ως κόμβοι. [26]

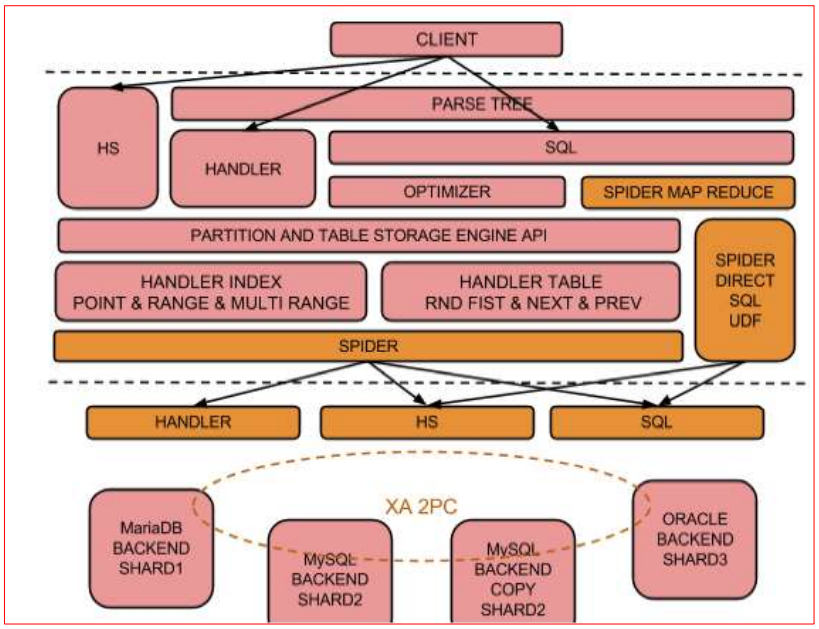
Οι κόμβοι που αποθηκεύουν τα δεδομένα θα σχεδιαστούν ως Backend κόμβοι και μπορεί να είναι στιγμιότυπα MariaDB, MySQL, Oracle χρησιμοποιώντας οποιαδήποτε μηχανή αποθήκευσης διαθέσιμη μέσα στο backend.

Οι κόμβοι Spider είναι στιγμιότυπα που τρέχουν τουλάχιστον το MariaDB 10.0.4. Χρησιμοποιούνται για να δηλώσουν ανά πίνακα σύνδεση στους κόμβους του backend. Επιπλέον, οι κόμβοι Spider μπορούν να εγκατασταθούν για να επιτρέψουν στους πίνακες να χωριστούν και να αντικατοπτρίζονται σε πολλαπλούς κόμβους Backend.

Το Spider είναι ένας επιπρόσθετος μηχανισμός αποθήκευσης δεδομένων, που ενεργεί ως υποκατάστατο μεταξύ του βελτιστοποιητή και των απομακρυσμένων backends. Όταν ο βελτιστοποιητής αιτείται πολλαπλές κλήσεις προς τη μηχανή αποθήκευσης, ο Spider επιβάλλει τη συνέπεια χρησιμοποιώντας το πρωτόκολλο 2 φάσεων δέσμευσης στα backends και δημιουργώντας συναλλαγές στα backends για να διατηρήσουν ατομικές λειτουργίες για μια ενιαία εκτέλεση SQL. Η διατήρηση της ατομικής λειτουργίας

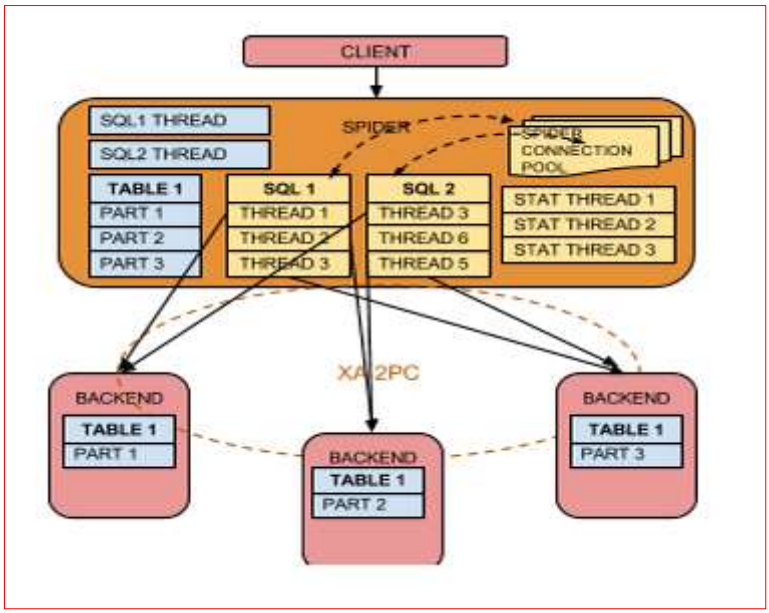
κατά την εκτέλεση χρησιμοποιείται σε πολλά επίπεδα της αρχιτεκτονικής. Για το κανονικό σχέδιο βελτιστοποίησης, αναφέρεται σε πολλαπλές αναγνώσεις και για ταυτόχρονες σαρώσεις κατατμήσεων, θα αναφέρεται σε ημι-συναλλαγές.

Τα δαπανηρά ερωτήματα μπορεί να είναι πιο αποτελεσματικά όταν είναι δυνατό να ωθηθεί πλήρως ένα μέρος του σχεδίου εκτέλεσης σε κάθε οπίσθιο τμήμα και να μειωθεί το συνολικό αποτέλεσμα αυτής της ενέργειας. Το Spider επιτρέπει την εκτέλεση με κάποιες συντομεύσεις άμεσης εκτέλεσης.



Εικόνα 20: Λειτουργία μηχανισμού αποθήκευσης δεδομένων Spider

Ο Spider χρησιμοποιεί τα μοντέλα ανά κατάτμηση και ανά πίνακα για ταυτόχρονη πρόσβαση στους κόμβους απομακρυσμένης βάσης δεδομένων. Για φορτίο μνήμης που μπορεί να χρησιμοποιηθεί για τον ορισμό πολλών κατατμήσεων σε έναν μοναδικό απομακρυσμένο κόμβο backend για την καλύτερη προσαρμογή της ταυτόχρονης πρόσβασης στις διαθέσιμες CPU στο υλισμικό.

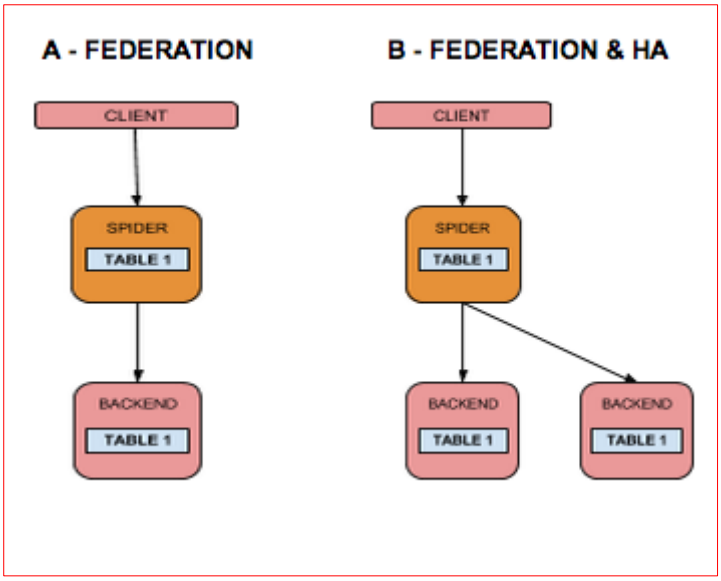


Εικόνα 21: Μοντέλο νημάτων spider

Ο Spider διατηρεί ένα εσωτερικό πίνακα , στον οποίο αποθηκεύει στατιστικά ευρετηρίου, που βασίζονται σε ξεχωριστά νήματα. Οι στατιστικές γίνονται ανά προεπιλογή με βάση τη χρονική γραμμή και αναφέρονται στο crd για την καρδιότητα (κατάσταση υγείας - cardinality) και τα sts (status) για την κατάσταση του πίνακα.

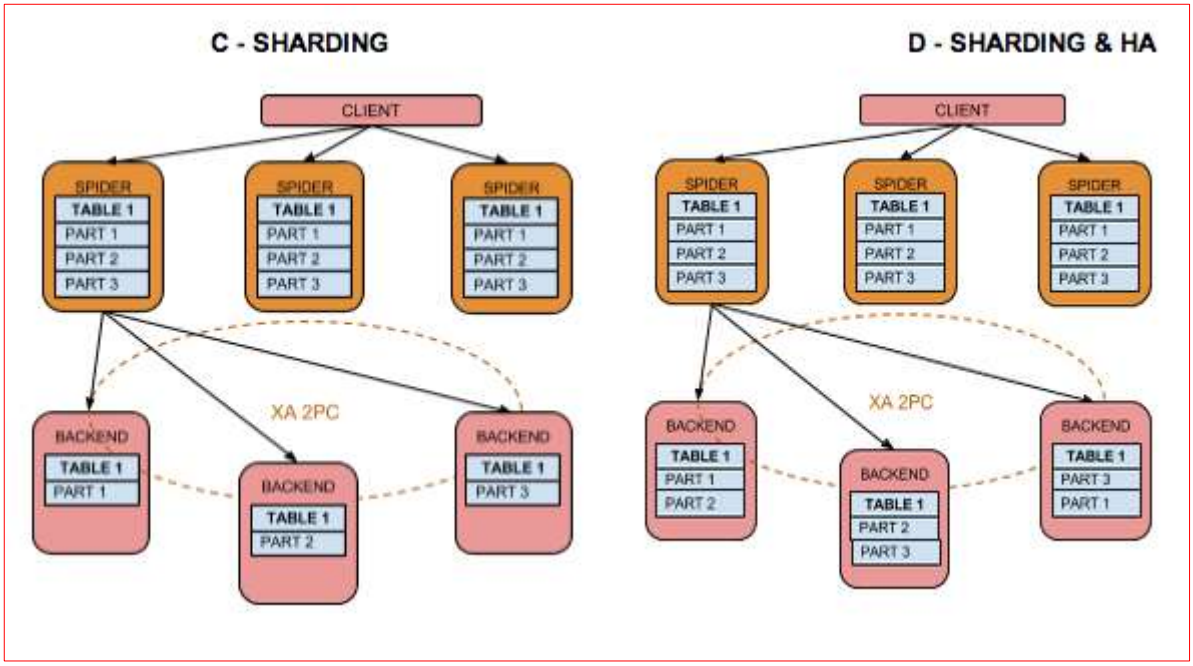
2.4.2 Συνήθης διατάξεις κόμβων spider.

Οι συνήθης διατάξεις κόμβων spider είναι οι επόμενες [25]:



Εικόνα 22: Διατάξεις κόμβων spider: A- Federation και B-Federation HA

- A – Federation: όπου υπάρχει ένας κόμβος backend με ένα πίνακα χωρίς κατάτμηση ο όποιος χαρτογραφείται στον spider κόμβο.
- B – Federation & HA: όπου υπάρχει ένα σύνολο από κόμβων backend τηρώντας αντίγραφο του ίδιου πίνακα με σκοπό την υψηλή διαθεσιμότητα. Ο κόμβος spider είναι συνδεδεμένος και στους δύο πίνακες .
- C – Sharding: όπου τα δεδομένα είναι οριζόντια κατακερματισμένα μεταξύ πολλαπλών κόμβων. Ένα σύνολο από κόμβους spider είναι συνδεδεμένοι στους προηγούμενους κόμβους, τηρώντας την συνολική εικόνα της βάσης .
- D – Sharding & HA: όπου τα δεδομένα είναι οριζόντια κατακερματισμένα μεταξύ πολλαπλών κόμβων και κάθε ένα από τους backend κόμβο τηρεί ένα τμήμα του επόμενου κόμβου. Ένα σύνολο από κόμβους spider είναι συνδεδεμένοι στους προηγούμενους κόμβους.



Εικόνα 23: Διατάξεις κόμβων spider: C-Sharding και D-Sharding HA

2.4.3 Σενάρια χρήσης πινάκων spider.

Θα πραγματοποιήσουμε παράδειγματα σεναρίων χρήσης των πινάκων spider. Για τον σκοπό αυτό θα αναφερθούμε σε τρεις εξυπηρετητές με την εξής ονοματολογία. [27]

- Spider: ο οποίος θα λειτουργεί ως frontend εξυπηρετητής, τηρώντας τον μηχανισμό αποθήκευσης spider.
- Backend1: ο οποίος θα λειτουργεί ως backend εξυπηρετητής, τηρώντας τα δεδομένα της βάσης.
- Backend2: ο οποίος θα λειτουργεί ως δεύτερος backend εξυπηρετητής, τηρώντας τα δεδομένα της βάσης.

Στον backend1 εξυπηρετητή και στον backend2.

Δημιουργούμε τον πίνακα στον οποίο και θα συνδέεται ο πίνακας spider.

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=InnoDB;
```

Εικόνα 24: Πίνακας στον backend1 και backend2

Ο προηγούμενος πίνακας θα είναι ίδιος και στους δύο backend εξυπηρετητές αλλά θα διατηρεί διαφορετικά δεδομένα πραγματοποιώντας οριζόντια κατάτμηση.

Στον εξυπηρετητή spider.

Καταγράφουμε τις πληροφορίες σύνδεσης για κάθε ένα εξυπηρετητή:

```
create server backend1 foreign data wrapper mysql options
(host '172.21.21.3', database 'test', user 'spider', password 'spider', port 3306);
create server backend2 foreign data wrapper mysql options
(host '172.21.21.4', database 'test', user 'spider', password 'spider', port 3306);
```

Εικόνα 25: Πίνακας συνδέσεων server στον spider

Τα πιθανά σενάρια πινάκων είναι τα εξής τέσσερα:

- Περίπτωση 1 : Απομακρυσμένος πίνακας.

Σε αυτήν την περίπτωση, δημιουργείται ένας πίνακας spider για να επιτρέπεται η απομακρυσμένη πρόσβαση στον πίνακα opportunities που φιλοξενείται στο backend1. Αυτό επιτρέπει στη συνέχεια τα ερωτήματα και την απευθείας σύνδεση στον διακομιστή: backend1 από τον κόμβο spider.

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=spider comment='wrapper "mysql", srv "backend1" table "opportunities"';
```

Εικόνα 26: Πίνακας spider περίπτωση χρήσης 1.

- Περίπτωση 2 : οριζόντιος κατακερματισμός μέσω hash.

Σε αυτή την περίπτωση δημιουργείται ένας πίνακας spider για τη διανομή δεδομένων μεταξύ του backend1 και του backend2, μέσω της στήλης id. Δεδομένου ότι η στήλη id είναι μια αυξανόμενη αριθμητική τιμή, ο κατακερματισμός θα διασφαλίσει την ομοιόμορφη κατανομή στους 2 κόμβους:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY HASH (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1"',
 PARTITION pt2 COMMENT = 'srv "backend2"'
) ;
```

Εικόνα 27: Πίνακας spider περίπτωση χρήσης 2.

Περίπτωση 3 : οριζόντιος κατακερματισμός κατά εύρος.

Σε αυτή την περίπτωση δημιουργείται ένας πίνακας spider για τη διανομή δεδομένων μεταξύ του backend1 και του backend2 με βάση το πρώτο γράμμα του πεδίου accountName. Όλα τα accountNames που ξεκινούν με το γράμμα L και τα προηγούμενα θα αποθηκευτούν στο backend1 και όλες τις άλλες τιμές που είναι

αποθηκευμένες στο backend2. Σημειώστε ότι η στήλη accountName πρέπει να προστεθεί στο πρωτεύον κλειδί που αποτελεί απαίτηση του καταμερισμού MariaDB:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id, accountName),
key(accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY range columns (accountName)
(
 PARTITION pt1 values less than ('M') COMMENT = 'srv "backend1"',
 PARTITION pt2 values less than (maxvalue) COMMENT = 'srv "backend2"'
) ;
```

Εικόνα 28: Πίνακας spider περίπτωση χρήσης 3

- Περίπτωση 4 : οριζόντιος κατακερματισμός κατά λίστα.

Σε αυτήν την περίπτωση δημιουργείται ένας πίνακας spider για τη διανομή δεδομένων μεταξύ του backend1 και backend2 με βάση συγκεκριμένες τιμές στο πεδίο owner. Οι Bill, Bob και Chris θα αποθηκευτούν στο backend1 και η Maria και Olivier θα αποθηκευτούν στο backend2. Σημειώστε ότι η στήλη ιδιοκτήτη πρέπει να προστεθεί στο πρωτεύον κλειδί που αποτελεί απαίτηση του διαχωρισμού MariaDB:

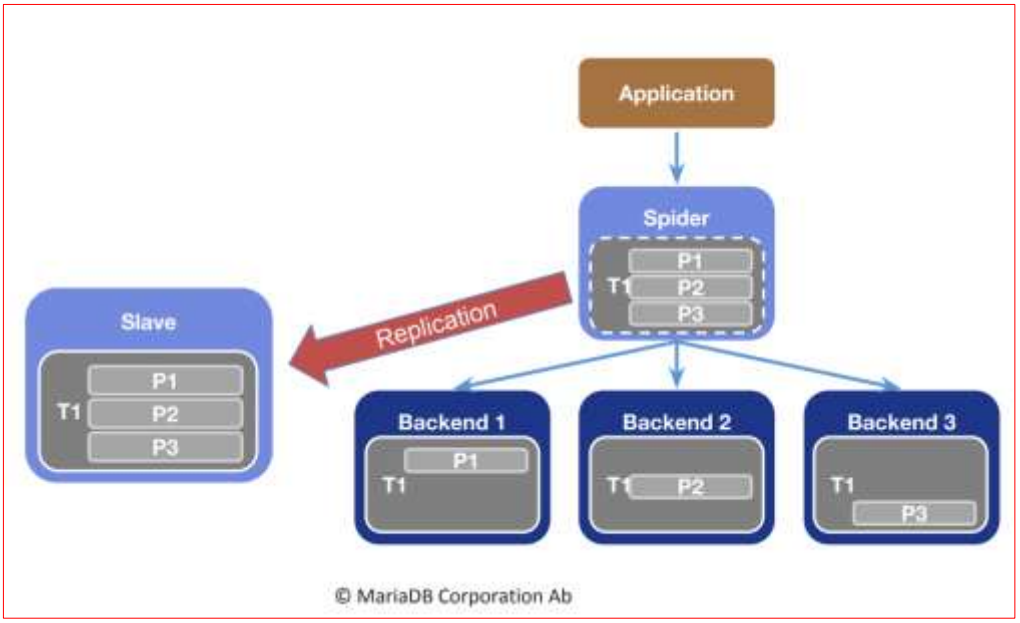
```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id, owner),
key(accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY list columns (owner)
(
 PARTITION pt1 values in ('Bill', 'Bob', 'Chris') COMMENT = 'srv "backend1"',
 PARTITION pt2 values in ('Maria', 'Olivier') COMMENT = 'srv "backend2"'
) ;
```

Εικόνα 29: Πίνακας spider περίπτωση χρήσης 4

2.4.4 Τήρηση αντιγράφων δεδομένων και spider.

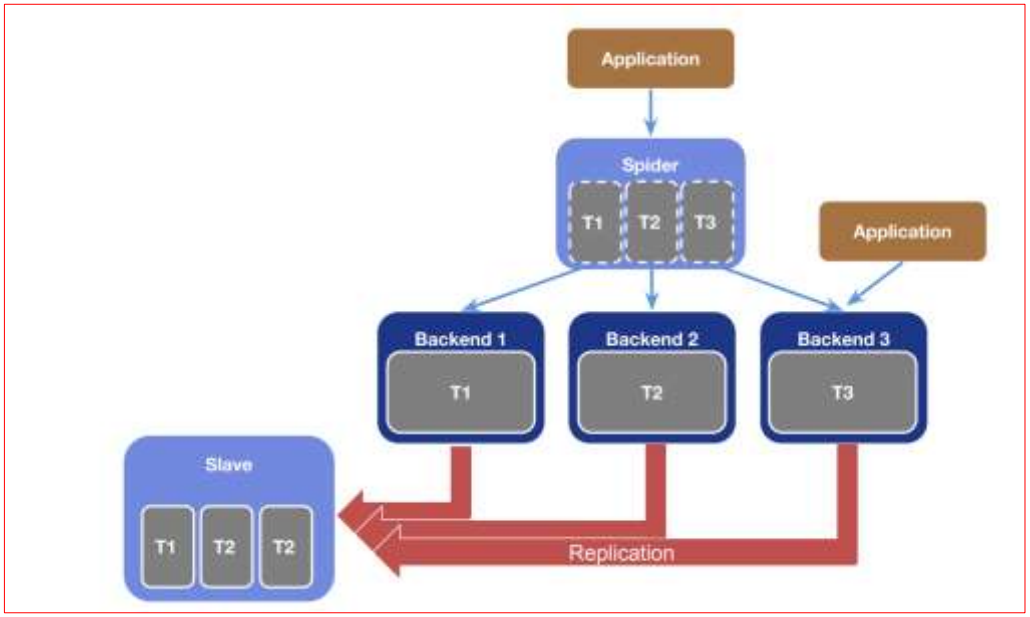
Τα δυνατά σενάρια χρήσης αντιγράφων ασφαλείας στην διάταξη των spider κόμβων είναι τα εξής δύο [23]:

- Καταγράφονται οι συναλλαγές που πραγματοποιούνται στην βάση δεδομένων τηρώντας μια διάταξη αφέντη – σκλάβου , όπου τον ρόλου του αφέντη επιτελεί ο spider κόμβος. Να υπενθυμίσουμε ότι ο spider κόμβος δεν τηρεί δεδομένα αλλά τα δεδομένα τηρούνται στους backend.



Εικόνα 30 : Διάταξη τήρησης αντιγράφων δεδομένων 1

- Πραγματοποιείται αντιγραφή των δεδομένων που τηρούνται στους backend απευθείας σε έναν κόμβο σκλάβο είτε τμηματικά είτε στο σύνολο των τμημάτων της βάσης.



Εικόνα 31 : Διάταξη τήρησης αντιγράφων 2

2.4.5 Πλεονεκτήματα χρήσης μηχανισμού αποθήκευσης δεδομένων spider.

Όταν ένας πίνακας που βασίζεται στον μηχανισμό spider δημιουργείται μία σύνδεση σε έναν πίνακα ενός απομακρυσμένου εξυπηρετητή δημιουργείται. [23],[28]

- Ο συνδεδεμένος πίνακας μπορεί να έχει όποια μηχανή αποθήκευσης δεδομένων επιθυμεί.
- Ο συνδεδεμένος πίνακας μπορεί να διαθέτει και ο ίδιος κατακερματισμό.
- Η ύπαρξη του πίνακα spider είναι διαφανής στον απομακρυσμένο πίνακα.
- Επιτρέπεται η ύπαρξη πολλαπλών πινάκων spider για τον ίδιο πίνακα.
- Επιτρέπει την αποκέντρωση μιας κατάτμησης σε έναν απομακρυσμένο εξυπηρετητή.
- Είναι ανεξάρτητη από την εφαρμογή.
- Πραγματοποιεί πολυπλεξία σε αντίγραφα χρησιμοποιώντας 2PC
- Αντιμετωπίζει το σύνδρομο της διάσπασης εγκεφάλου μέσω της συγκατάθεσης της πλειοψηφίας.
- Μπορεί να χρησιμοποιήσει τεχνικές τήρησης αντιγράφων μέσω της αντιγραφής των δεδομένων είτε των κατατμήσεων είτε του πίνακα spider.
- Διαθέτει διατάξεις υψηλής διαθεσιμότητας όταν βρίσκεται σε μια συστάδα εξυπηρετητών Γαλέρα.

2.5 Μετροπρόγραμμα SysBench

Το SysBench είναι ένα αρθρωτό εργαλείο, μεταξύ διαφορετικών πλατφόρμων και πολλαπλών νημάτων που χρησιμοποιείται με σκοπό την συγκριτική αξιολόγηση παραμέτρων λειτουργικών συστημάτων μια βάσης δεδομένων κάτω από εκτεταμένο φορτίο. [29]

Το SysBench επιτρέπει την εκτέλεση δοκιμής των ακόλουθων παραμέτρων του συστήματος:

- την απόδοση εισόδου / εξόδου αρχείων
- απόδοση προγραμματιστή.
- κατανομή μνήμης και ταχύτητα μεταφοράς
- απόδοσης υλοποίησης των νημάτων POSIX
- απόδοσης του διακομιστή βάσης δεδομένων

Το SysBench εκτελεί έναν καθορισμένο αριθμό νημάτων τα οποία εκτελούν παράλληλα αιτήσεις. Ο πραγματικός φόρτος εργασίας που παράγεται από τις αιτήσεις εξαρτάται από τη συγκεκριμένη δοκιμασία που πραγματοποιείται.

Αυτή την στιγμή βρίσκεται στην έκδοση 1.0.8 και μπορεί να γίνει εγκατάσταση από το αποθετήριο της percona. [30]

```
[root@10 ~]# sysbench --db-driver=mysql --oltp-table-size=3500000 --mysql-port=3306 --mysql-
user=sbuser --mysql-db=backend --mysql-password=1234 /usr/share/sysbench/tests/include/oltp_
legacy/parallel_prepare.lua prepare
sysbench 1.0.8 (using bundled LuaJIT 2.1.0-beta2)

Creating table 'sbtest1'...
Inserting 3500000 records into 'sbtest1'
Creating secondary indexes on 'sbtest1'...
```

Εικόνα 32 : Παράδειγμα δοκιμασίας δημιουργίας 3.500.000 εγγραφών μέσω του SysBench.

Στην εξομοίωση μας το SysBench χρησιμοποιήθηκε για την δημιουργία ενός πίνακα 7.000.000 εγγραφών σε διάφορους κόμβους με σκοπό την εξαγωγή συμπερασμάτων.

Ο πίνακας που δημιουργήθηκε είναι ο sbtest , ο οποίος είναι και ο προεπιλεγμένος πίνακας που θα δημιουργηθεί και παρουσιάζεται παρακάτω.

```
CREATE TABLE `sbtest` (
`id` int(10) unsigned NOT NULL auto_increment,
`k` int(10) unsigned NOT NULL default '0',
`c` char(120) NOT NULL default '',
`pad` char(60) NOT NULL default '',
PRIMARY KEY  (`id`),
KEY `k` (`k`));
```

Εικόνα 33: Πίνακας sbtest

Ο πίνακας spider που θα χρησιμοποιηθεί για την σύνδεση με τον πίνακα sbtest θα είναι ο επόμενος στον οποίο και έχουμε πραγματοποιήσει κατακερματισμό κατά εύρος τιμών και θα τον αναλύσουμε στο τελευταίο κεφάλαιο, όπου θα πραγματοποιηθούν δοκιμασίες σε αυτόν.

.

```
ariaDB [backend]> CREATE TABLE backend.sbtest1
    -> (
    ->   id int(10) unsigned NOT NULL AUTO_INCREMENT,
    ->   k int(10) unsigned NOT NULL DEFAULT '0',
    ->   c char(120) NOT NULL DEFAULT '',
    ->   pad char(60) NOT NULL DEFAULT '',
    ->   PRIMARY KEY (id),
    ->   KEY k (k)
    -> )
    -> ENGINE=SPIDER COMMENT='user "sbuser", password
    '> "1234", port "3306", table "sbtest1"'
    -> PARTITION BY RANGE(id)
    -> (
    -> PARTITION p1 VALUES LESS THAN (3500000)
    -> COMMENT 'host "192.168.56.233"',
    -> PARTITION p2 VALUES LESS THAN MAXVALUE
    -> COMMENT 'host "192.168.56.232"'
    -> );
uery OK, 0 rows affected (0.00 sec)
```

Εικόνα 34: Υλοποίηση πίνακα sbtest σε spider.

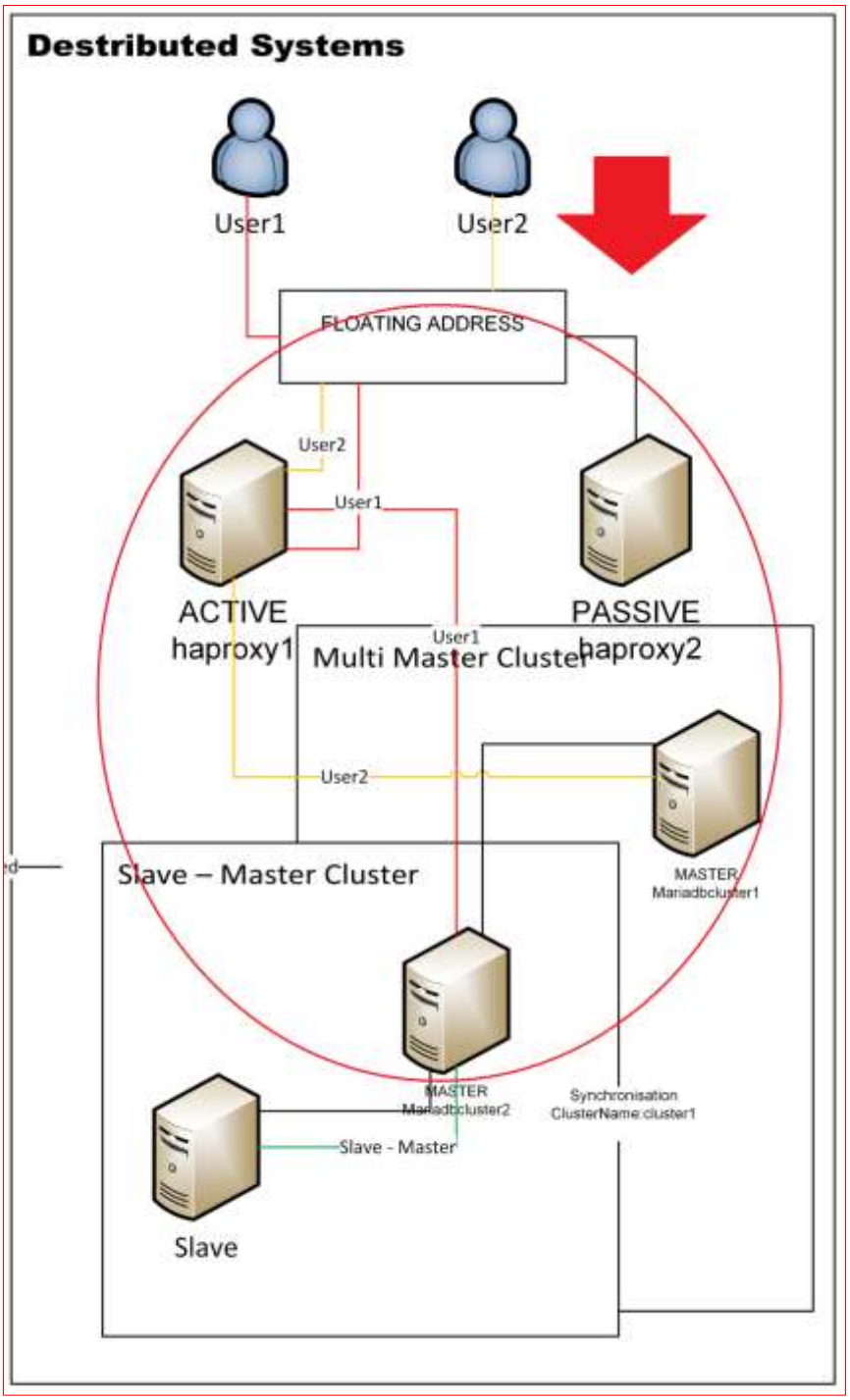
3. ΠΕΡΙΓΡΑΦΗ ΠΕΙΡΑΜΑΤΟΣ

3.1 Εισαγωγή – Συνοπτική παρουσίαση του πειράματος.

Το πείραμά μας αποτελείται από τρία επιμέρους τμήματα τα οποία και θα αναλύσουμε στις παρακάτω ενότητες.

3.1.1 Υψηλής διαθεσιμότητας συστάδα εξυπηρετητών πολλαπλών αφεντών.

Στο πρώτο τμήμα θα δημιουργηθεί μια υψηλής διαθεσιμότητας συστάδα εξυπηρετητών πολλαπλών αφεντών .Αυτή η διάταξη θα προσομοιάσει την ύπαρξη ενός κέντρου δεδομένων (datacenter).



Εικόνα 35 : Υψηλής διαθεσιμότητας συστάδα εξυπηρετητών.

Οι κόμβοι της βάσης θα βρίσκονται σε διάταξη πολλαπλών αφεντών. Θα διαθέτουν το ίδιο τμήμα της βάσης δεδομένων ενώ όλα τα αιτήματα προς αυτή θα ικανοποιούνται με μία διεύθυνση IP, την VIP. Ο αριθμός των κόμβων της βάσης θα πρέπει να είναι περιττός αριθμός για να υπάρχει η συγκατάθεση της πλειοψηφίας . Όλα τα αιτήματα εγγραφών προς την βάση θα πρέπει να γίνουν αποδεκτές από το σύνολο των κόμβων πριν εφαρμοστούν σε αυτή. Σε περίπτωση που υπάρχει σύγκρουση στο προηγούμενο τηρείται η εκδοχή της πλειοψηφίας των κόμβων.

Μπροστά από το σύνολο των κόμβων, που διαθέτουν βάσεις, θα υπάρχουν ένα σύνολο από άνω των δύο HAProxy. Ένας κύριος HAProxy θα επιτηρεί την συστάδα και θα επιτελεί το ρόλο του εξισορροπιστή φορτίου κατευθύνοντας τα αιτήματα προς την βάση δεδομένων. Οι υπόλοιποι HAProxy θα τηρούνται σε ετοιμότητα να αναλάβουν τον ρόλο του κύριου HAProxy σε περίπτωση μη διαθεσιμότητας του προηγούμενου. Σε περίπτωση μη διαθεσιμότητας κάποιου κόμβους της βάσης , ο κύριος HAProxy ενημερώνεται μέσω των ελέγχων υγείας και διακόπτει τα αιτήματα προς αυτόν τον κόμβο.

Το σύνολο της συστάδας εξυπηρετητών θα διαθέτει μια διεύθυνση IP, την VIP, η οποία και θα εναλλάσσεται μεταξύ των κύριων HAProxy επιτρέποντας υψηλή διαθεσιμότητα της συστάδας, σε περίπτωση μη διαθεσιμότητας ένα από τους κόμβους HAProxy Αυτό θα έχει σαν αποτέλεσμα οι συναλλαγές των πελατών προς την βάση να είναι διαφανής, δίνοντας την αίσθηση στον τελικό χρήστη της ύπαρξης μιας κεντρικοποιημένης βάσης παρά μια κατανεμημένης, ικανοποιώντας με αυτό τον τρόπο τους κανόνες του Date.

Οι τεχνολογίες που θα χρησιμοποιηθούν είναι οι επόμενες:

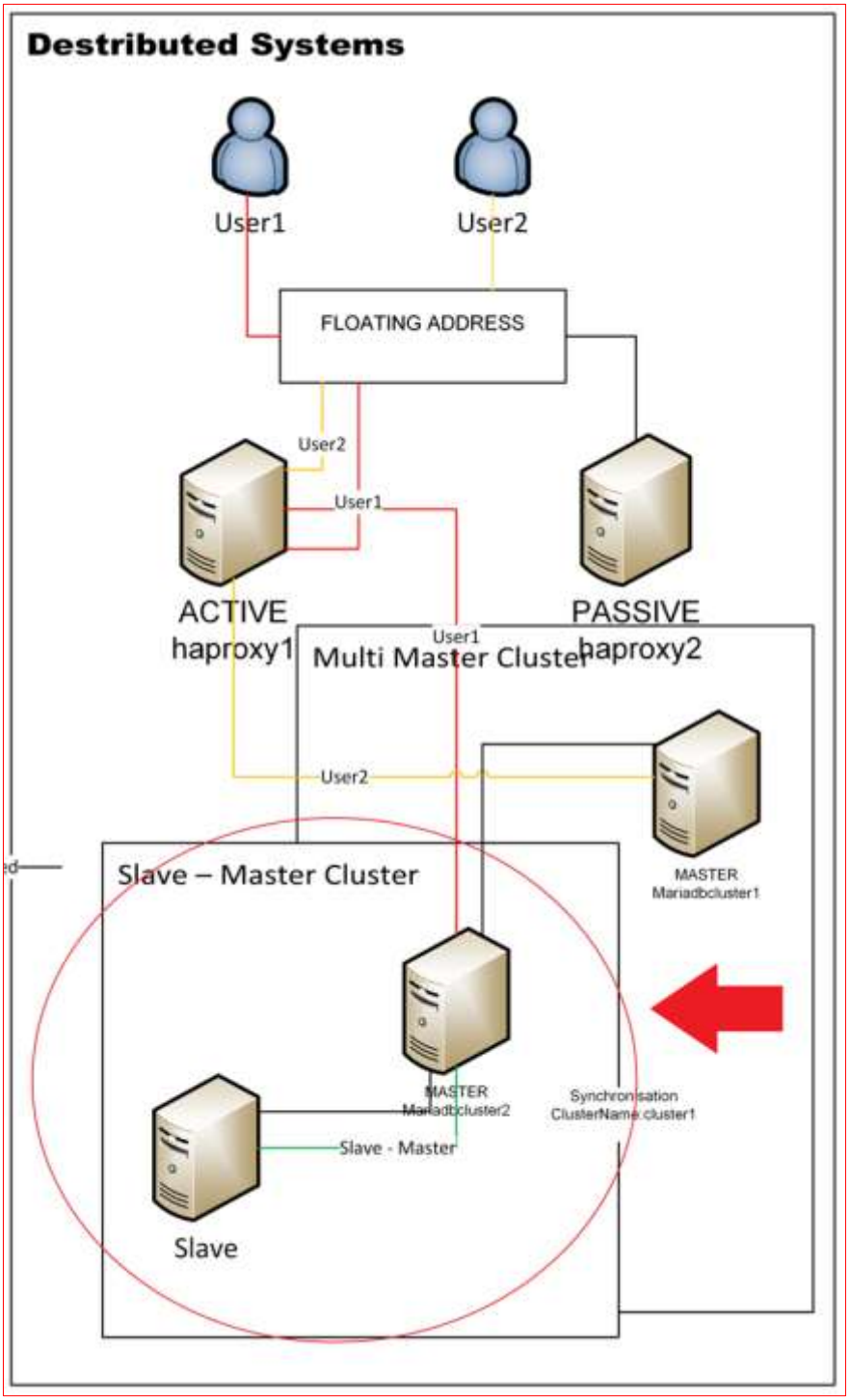
- Συστάδα εξυπηρετητών Γαλέρα – Βάση Δεδομένων.
- Εξυπηρετητές HAProxy – Εξισορρόπηση Φορτίου.
- Το VIP – Keepalived.

3.1.2 Διάταξη αντιγραφής δεδομένων αφέντη – σκλάβου.

Στο δεύτερο τμήμα θα δημιουργηθεί μια διάταξη κόμβων αφέντη – σκλάβου σε κάθε ένα κέντρο δεδομένων. Ο σκοπός που θα επιτευθεί με το προηγούμενο θα είναι η ύπαρξη αντιγράφων ασφάλειας σε περίπτωση καταστροφικής αποτυχίας των κόμβων, που τηρούν τα δεδομένα της βάσης.

Η προτιμητέα λύση θα ήταν ο κόμβος σκλάβος να βρίσκεται σε κάποια εγκατάσταση διαφορετική από το κέντρο δεδομένων, έτσι ώστε σε περίπτωση ακόμα και φυσική καταστροφής να υπάρχει δυνατότητα τήρησης αντιγράφων ασφαλείας.

Τήρηση αντιγράφων ασφαλείας μπορεί να πραγματοποιηθεί και στον κόμβο spider. Ο κόμβος spider δεν διαθέτει τα πραγματικά δεδομένα, αλλά μια χαρτογράφηση των προηγουμένων που βρίσκονται στους κόμβους των βάσεων δεδομένων Backend. Αυτό σημαίνει ότι σε περίπτωση μη διαθεσιμότητας κάποιο κόμβου βάσης Backend , ο spider δεν μπορεί να χαρτογραφήσει αυτά τα δεδομένα, αφού δεν υπάρχει σύνδεση. Η διάταξη όμως αφέντη – σκλάβου φέρνει την λύση σε αυτό το πρόβλημα καθώς τηρεί το σύνολο των συναλλαγών που πραγματοποιούνται στον κόμβο spider , με αποτέλεσμα οι εγγραφές που χαρτογραφούνται στο προηγούμενο κόμβο να εισάγονται ως εγγραφές στον κόμβο σκλάβο του spider. Αποτέλεσμα του προηγούμενου είναι η δυνατότητα πρόσβασης σε αυτά σε περίπτωση αδυναμίας σύνδεση με τον κόμβο backend. Στο πείραμα που πραγματοποιήσαμε προσομοιώσαμε αυτή την κατάσταση.

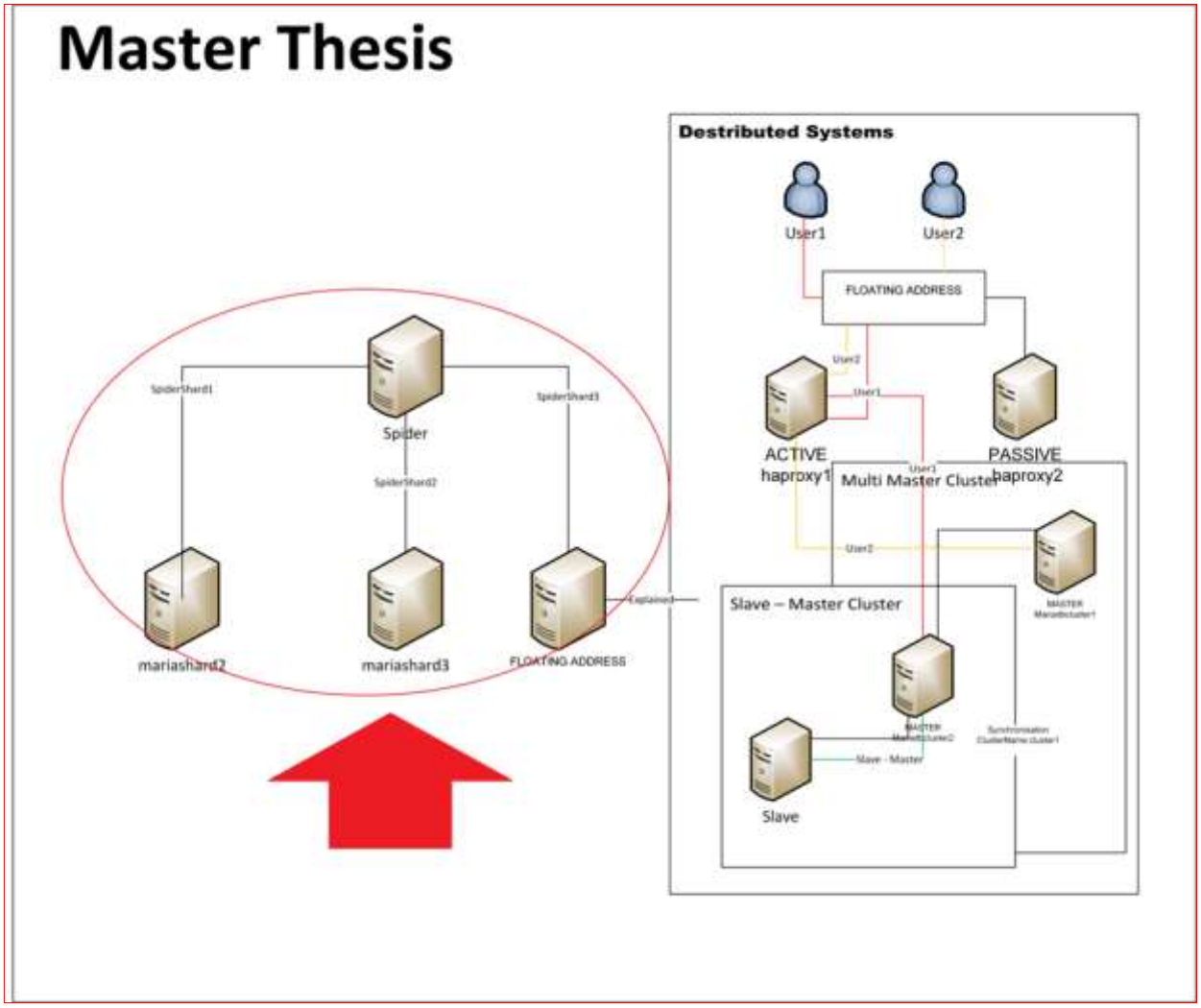


Εικόνα 36 : Διάταξη αφέντη - σκλάβου

3.1.3 Οριζόντια κατακερμάτιση βάσης δεδομένων μέσω της μηχανής αποθήκευσης spider.

Στο τρίτο τμήμα θα προσομοιαστεί η δημιουργία μιας οριζόντιας κατακερματισμένης βάσης δεδομένων μεταξύ διαφορετικών κέντρων δεδομένων. Τα κέντρα δεδομένων θα τηρούν την διάταξη υψηλής διαθεσιμότητας που παρουσιάσαμε στο πρώτο τμήμα και δεν θα γνωρίζουν για την ύπαρξη του κόμβου spider δίνοντας διαφάνεια στην διάταξή μας. Ένα ακόμα πλεονέκτημα που θα παρατηρήσουμε θα είναι η κλιμάκωση της βάσης δεδομένων μας σε ένα σύνολο κόμβων και τα αποτελέσματα στο χρόνο εκτέλεσης των

συναλλαγών, με την διαδικασία κατακερματισμού των δεδομένων σε ένα σύνολο κόμβων, σε αντίθεση με την τήρησή τους σε έναν και μόνο κόμβο (εντατικοποιημένη διάταξη βάσης δεδομένων.)



Εικόνα 37: Συνολική διάταξη της οριζόντια κατακερματισμένης βάσης δεδομένων μας

3.2 Υλοποίηση του πειράματός μας.

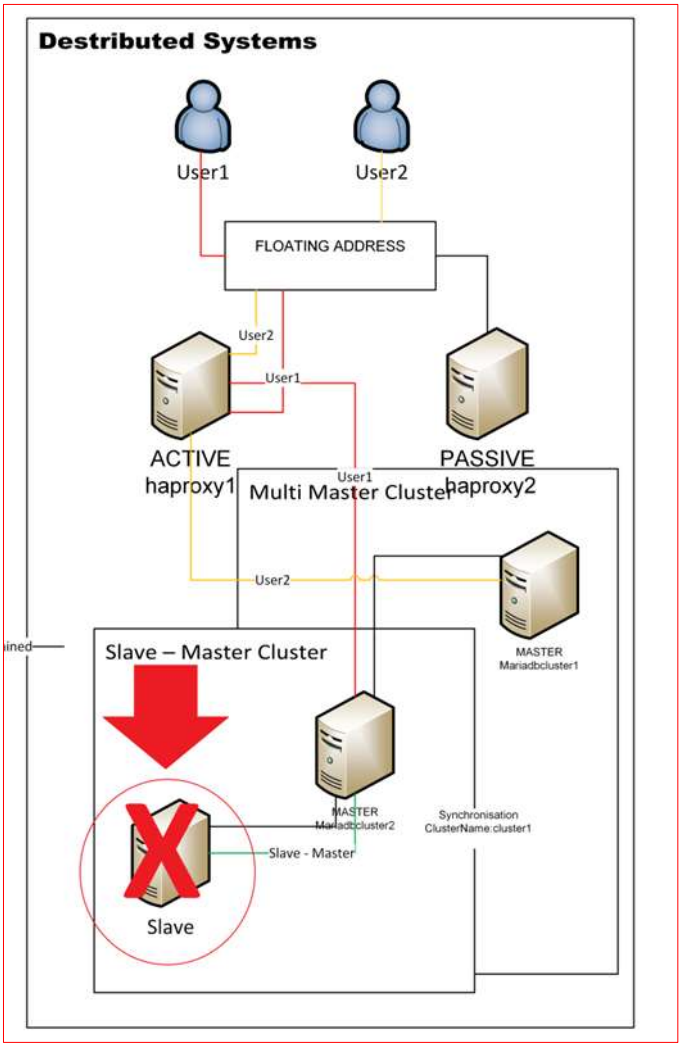
Στην παρούσα ενότητα θα αναλυθεί η υλοποίηση του πειράματος που πραγματοποιήσαμε.

3.2.1 Υλοποίηση της διάταξης υψηλής διαθεσιμότητας

3.2.1.1 Σκοπός

Θα δημιουργήσουμε 2 κόμβους HAProxy μπροστά από 2 κόμβους της βάσης δεδομένων μας καθώς και μια VIP ως εξής:

- VIP: 192.168.56.227
- Κόμβος1- haproxy1: 192.168.56.225
- Κόμβος2- haproxy2: 192.168.56.226
- Κόμβος3- mariadbcluster1: 192.168.56.221
- Κόμβος4- mariadbcluster3: 192.168.56.222



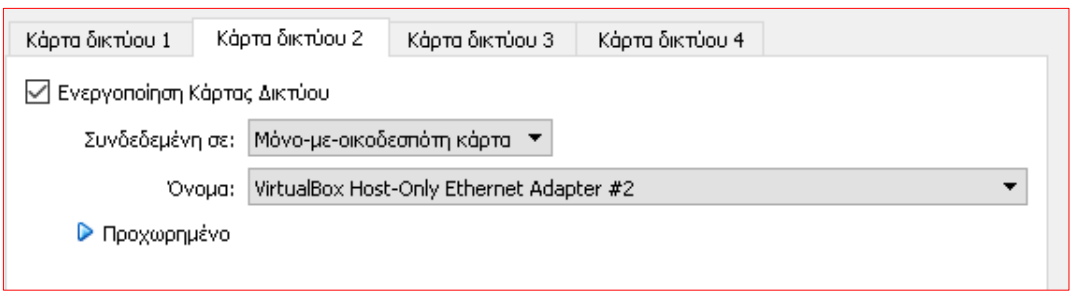
Εικόνα 38. Πείραμα με HAProxy - VIP

Οι κόμβοι, θα προσομοιάζουν ένα κέντρο δεδομένων. Δεν θα υλοποιηθεί η διάταξη αφέντη – σκλάβου που θα πραγματοποιηθεί σε επόμενη διάταξη.

3.2.1.2 Προαπαιτούμενα

Στο πείραμά μας χρησιμοποιήσαμε το Oracle VM VirtualBox .Για την δικτύωση των εικονικών μηχανών (virtual machine- VM) χρησιμοποιήθηκε το VirtualBox Host-Only Adapter. Φυσικά οφείλουμε να το έχουμε ρυθμίσει σε κάθε ένα VM στις ρυθμίσεις δικτύου κάθε VM.

Εικόνα 39:VirtualBox Host-Only Adapter



Εικόνα 40 : Ρυθμίσεις Δικτύου VM

Σε κάθε ένα VM έχουμε παραχωρήσει 1GB RAM και 8 GB HDD. Το λειτουργικό που χρησιμοποιήθηκε είναι το centOs7 και συγκεκριμένα το CentOS-7-x86_64-Minimal-1611 edition. Για την εγκατάσταση των VMS στο VirtualBox χρησιμοποιήσαμε της οδηγίες που μας παρείχε η ιστοσελίδα της MariaDB.[31]

Πραγματοποιήσαμε εγκατάσταση του MariaDB 10.0 από τα αποθετήρια της MariaDB σε όλα τα VMs και δημιουργήσαμε τον χρήστη cluster-user ο οποίος και θα χρησιμοποιηθεί σε όλα μας τα πειράματα.

Εικόνα 41. HA keepalived cluster

3.2.1.3 Εγκατάσταση Galera Cluser

Πραγματοποιήσαμε ρυθμίσεις στο αρχείο /etc/my.cnf.d/server.cnf των κόμβων που θα συμμετέχουν στην συστάδα καθορίζοντας ποιοι κόμβοι θα συμμετέχουν σε αυτήν.

```
 GNU nano 2.3.1            File: /etc/my.cnf.d/server.cnf


bind-address=0.0.0.0

#
# * Galera-related settings
#
[galera]
query_cache_size=0
binlog_format=ROW
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
wsrep_provider=/usr/lib64/galera/libgalera_smm.so
wsrep_cluster_address="gcomm://192.168.56.221,192.168.56.222,192.168.56.223"
wsrep_cluster_name='cluster1'
wsrep_node_address='192.168.56.221'
wsrep_node_name='mariadbcluster1'
wsrep_sst_method=rsync
wsrep_sst_auth=cluster-user:clusterpass
```

Εικόνα 42: Ρυθμίσεις στο αρχείο /etc/my.cnf.d/server.cnf

Ακολουθώντας της οδηγίες που μας παρείχε η ιστοσελίδα της Galera Cluster, εκκινήσαμε τους δύο κόμβους που θα συμμετέχουν στην συστάδα.[32]

Οι εντολές που χρησιμοποιήθηκαν είναι ακόλουθες:

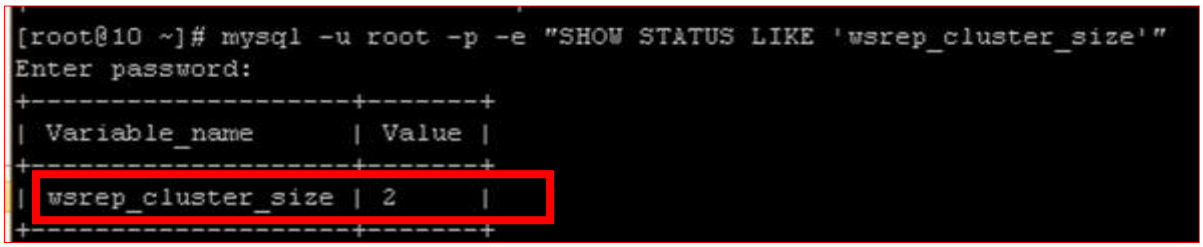
- /etc/init.d/mysql bootstrap (στον πρώτο κόμβο)
- /etc/init.d/mysql start (σε όλους τους άλλους κόμβους)

Καθορίσαμε με την προηγούμενη διαδικασία ότι ο πρώτος κόμβος , ο οποίος θα είναι και ο πρώτος που θα λειτουργήσει θα διαθέτει το πρωτεύων τμήμα. Όταν ξεκινά η συστάδα δεν υπάρχει πρωτεύων τμήμα, για να αρχικοποιηθεί θα πρέπει να καθορίσουμε έναν κόμβο να το κάνει με το να είναι ο πρώτος κόμβος που θα εισαχθεί στην συστάδα και θα γίνει λειτουργικός. Έπειτα κάθε ένας καινούργιος κόμβος που θα εκκινεί με σκοπό να συνδεθεί στην συστάδα, θα επιδιώκει να αποκτήσει σύνδεση δικτύου με τους άλλους κόμβους στην συστάδα εξυπηρετητών. Για κάθε έναν κόμβο που βρίσκουν, ελέγχουν άμα ανήκει ή όχι στο τμήμα του πρωτεύοντος τμήματος .Όταν βρουν το πρωτεύων τμήμα, κάνουν αίτηση για μεταφορά κατάστασης (SST) για να συγχρονίσουν την τοπική τους βάση με αυτή της συστάδας.

.

Όταν ένας κόμβος που είχε διακόψει την σύνδεση επιστρέφει στην συστάδα, λαμβάνει όλες τις αλλαγές , πραγματοποιώντας αύξουσα μεταφορά κατάστασης (IST) και σαν αποτέλεσμα κάνει το περιεχόμενο του πρόσφατο. Άμα το προηγούμενο δεν είναι δυνατό πραγματοποιεί μια ολική μεταφορά SST. Στην περίπτωση που δεν πραγματοποιηθεί κάτι από τα προηγούμενα δεν γίνεται αποδεκτός από την συστάδα και παραμένει μη λειτουργικός.

Σε περίπτωση επιτυχίας της προηγούμενης δραστηριότητας ,πραγματοποιώντας ένα ερώτημα στην βάση μας σχετικά με τον αριθμό των μελών σε αυτή, θα πρέπει να μας επιστρέψει όσους κόμβους έχουμε εισάγει. Στην προκειμένη περίπτωση αυτοί είναι δύο. Είναι σημαντικό να αναφέρουμε σε αυτό το σημείο πως ο αριθμός των κόμβων της βάσης θα πρέπει πάντα να είναι περιττός αριθμός. Ο λόγος για το προηγούμενο είναι για να επιτευθεί η συγκατάθεση της πλειοψηφίας σε περίπτωση σύγκρουσης δεδομένων.



Εικόνα 43 : Επιτυχές αποτέλεσμα δημιουργίας συστάδας με δύο εξυπηρετητές

3.2.1.4 Εγκατάσταση HAProxy - Keepalived - Xinetd

Αφού πραγματοποιηθεί η εγκατάσταση της συστάδας εξυπηρετητών Γαλέρα, οι οποίοι και θα διαθέτουν την βάση μας θα ακολουθεί η δημιουργία των δυο HAProxy που θα έχουν τον ρόλο του εξισορροπιστή φορτίου.

Μετά την εγκατάσταση των xinetd, harpoxy και keealived θα πραγματοποιήσουμε ρυθμίσεις στα αρχεία :

- /usr/bin/clustercheck
- /etc/xinetd.d/mysqlchk
- /etc/haproxy/haproxy.cfg.
- /etc/keepalived/keepalived.conf

Στο πρώτο θα καθοριστούν οι συνθήκες για τον έλεγχος της υγείας των βάσεων δεδομένων, που βρίσκονται στο πίσω μέρος της διάταξής μας και θα το πραγματοποιήσει ο δαίμονας xinetd.

Στο δεύτερο θα οριστούν οι αρχές που θα χρησιμοποιηθούν για την εξισορρόπηση του φορτίου μεταξύ των διαφορετικών κόμβων της βάσης από τους HAProxy ,που βρίσκονται στο μπροστά μέρος της διάταξής μας. Κύριο ρόλο θα διαδραματίσει ο αλγόριθμος εξισορρόπησης φορτίου, που σε αυτό το πείραμα έχει τεθεί στο roundrobin.

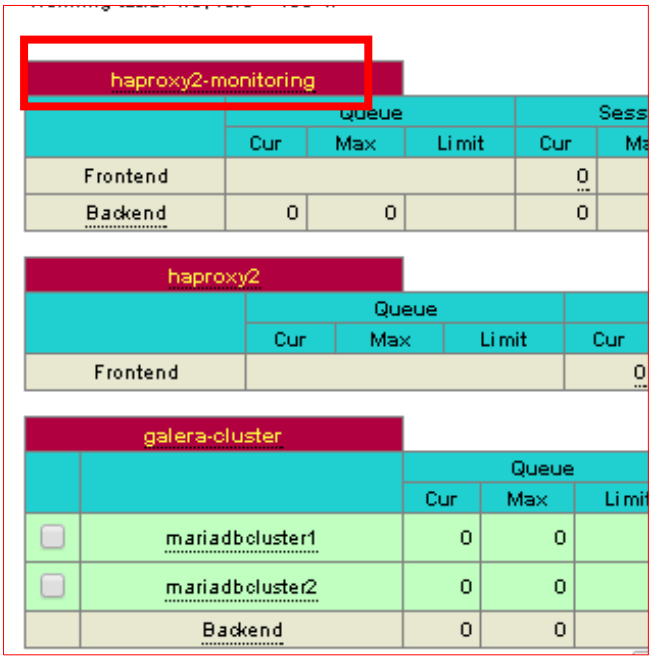
Στο τρίτο θα οριστεί οι παράμετροι για την προτεραιότητα στην κατοχή της VIP μεταξύ των κόμβων HAProxy, κύριο ρόλο θα πραγματοποιήσουν τα πακέτα VRRP.

Λεπτομερείς αναφορά για τα προηγούμενα καθώς και τα αρχεία κώδικα που χρησιμοποιήθηκαν έχει πραγματοποιηθεί στο κεφάλαιο 2 του παρόντος.

Μετά την πραγματοποιήσει των προηγούμενων θα ακολουθήσει η ενεργοποίηση των xinetd , HAProxy καθώς και του keepalived με τις εντολές:

- sudo systemctl start xinetd
- sudo systemctl start haproxy
- sudo systemctl start keepalived

Επιτυχής εκτέλεση των προηγούμενων θα έχεις ως αποτέλεσμα να έχουμε πρόσβαση στην σελίδα των στατιστικών του HAProxy εισάγοντας σε ένα φυλλομετρητή το http://192.168.56.227/ η οποία είναι η διεύθυνση της VIP καθώς και τα διαπιστευτήρια “username” και password”.



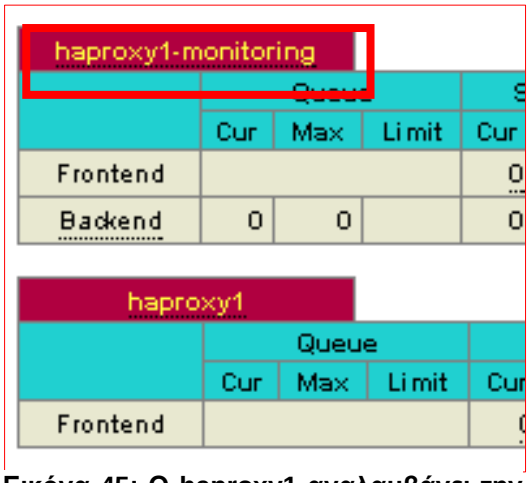
Εικόνα 44: Καρτέλα στατιστικών HAProxy

3.2.1.5 Έλεγχος επιτυχής λειτουργίας της διάταξής μας.

3.2.1.5.1 Θέτουμε τον haproxy2 σε κατάσταση μη διαθεσιμότητας πραγματοποιώντας τερματισμό αυτού.

Ο haproxy1 αναμένει να ακούει πακέτα διαφήμισης VRRP πολλαπλής διανομής, τα οποία ο haproxy2 εκπέμπει σε τακτά χρονικά διαστήματα. Το προεπιλεγμένο διάστημα διαφήμισης είναι ένα δευτερόλεπτο. Μετά την διέλευση τριών διαδοχικών διαφημίσεων VRRP, ο haproxy1 αναλαμβάνει την VIP και μαζί με αυτό την εξισορρόπηση του φορτίου στην σύσταδα μας.

Το αποτέλεσμα της προηγούμενης ενέργεια είναι το κάτωθι.



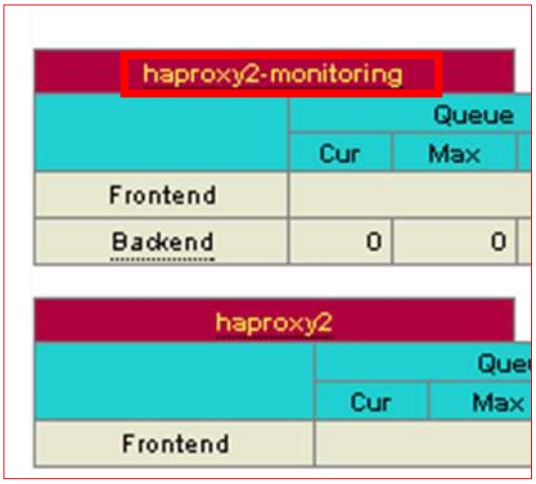
Εικόνα 45: Ο haproxy1 αναλαμβάνει την VIP

3.2.1.5.2 **Θέτουμε απευθείας στον haproxy2, ότι είναι ο keepalived.**

Στον haproxy2 εισάγουμε την παρακάτω εντολή :

- systemctl restart keepalived

Το επιτυχές αποτέλεσμα του προηγούμενου είναι ο haproxy2 να αναλαμβάνει την VIP.



Εικόνα 46: Ο haproxy2 αναλαμβάνει την VIP

3.2.1.5.3 **Δοκιμή του clustercheck τοπικά στο 192.168.56.221 ->mariadbcluster1**

Στο κόμβο mariadbcluster1 που διαθέτει την βάση εκτελούμε την κάτωθι εντολή και αναμένουμε τον κώδικα απόκρισης HTTP 200 OK:

- /usr/bin/clustercheck

```
[root@10 ~]# sudo /usr/bin/clustercheck
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 40
```

Εικόνα 47: Clustercheck mariadbcluster1

Αυτό το αρχείο κώδικα θα εκτελείται από μια υπηρεσία, την mysqlchk του δαίμονα (daemon) xinetd (extended internet daemon). Ο δαίμονας θα έχει σαν αποστολή να ακούει μια συγκεκριμένη θύρα (σαν αρχική επιλογή είναι η 9200).

3.2.1.5.4 **Δοκιμή του clustercheck τοπικά στο 192.168.56.222 ->mariadbcluster2**

Στο κόμβο mariadbcluster2 που διαθέτει την βάση εκτελούμε την κάτωθι εντολή και αναμένουμε τον κώδικα απόκρισης HTTP 200 OK:

- /usr/bin/clustercheck

```
[root@10 ~]# sudo /usr/bin/clustercheck
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 40

Percona XtraDB Cluster Node is synced.
```

Εικόνα 48:Clustercheck mariadbcluster2

3.2.1.5.5 **Απομακρυσμένα στο 192.168.56.221 ->mariadbcluster1 από τον haproxy1**

Στον κόμβο haproxy1 εκτελούμε την κάτωθι εντολή και αναμένουμε τον κώδικα απόκρισης HTTP 200 OK:

- telnet 192.168.56.221 9200

```
[root@10 ~]# telnet 192.168.56.221 9200
Trying 192.168.56.221...
Connected to 192.168.56.221.
HTTP/1.1 200 OK
Connection: close
Content-Length: 40

Percona XtraDB Cluster Node is synced.
Connection closed by foreign host.
```

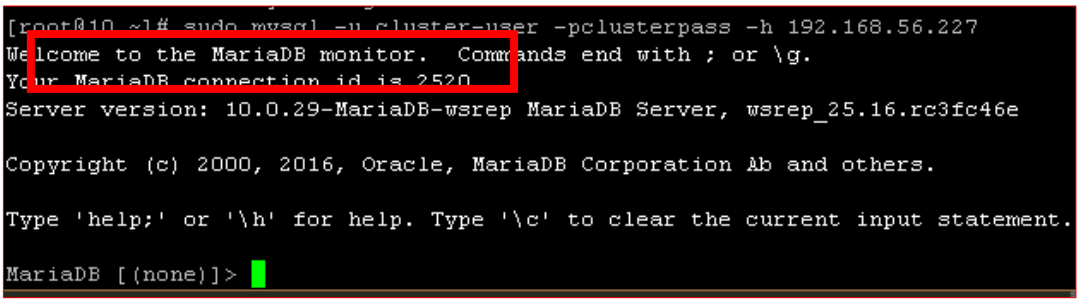
Εικόνα 49 :Telnet haproxy1

3.2.1.5.6 **Συνδεόμαστε στην βάση δεδομένων μας που βρίσκεται στην συστάδα εξυπηρετητών Γαλέρα μέσω της VIP (Virtual IP ή Floating IP) 192.168.56.227**

Στον κόμβο haproxy2 εκτελούμε την κάτωθι εντολή και αναμένουμε να συνδεθούμε με την βάση:

- sudo mysql -u cluster-user -pclusterpass -h 192.168.56.227

Το επιτυχές αποτέλεσμα θα μοιάζει με το επόμενο, οπότε έχουμε αποκτήσει πρόσβαση στην βάσεων δεδομένων μας, που ήταν και ο επιθυμητός σκοπός του όλου εγχειρήματος και έχουμε ως αναγνωριστικό σύνδεσης 2520.



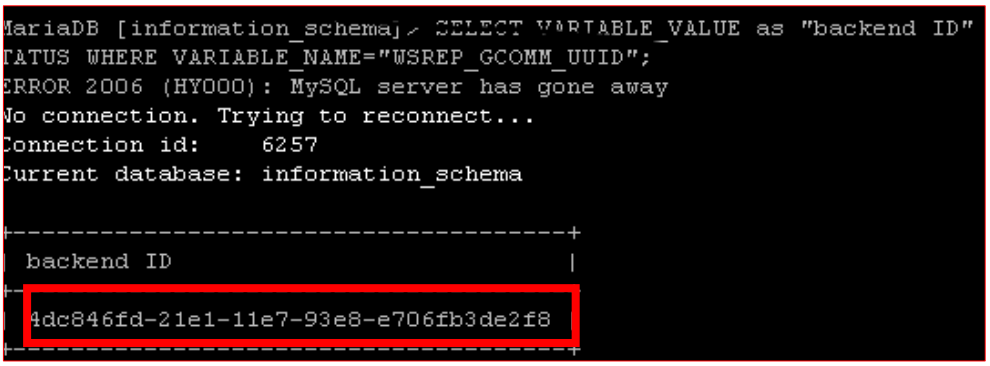
Εικόνα 50: Επιτυχής σύνδεση με την βάση δεδομένων μας μέσω VIP.

3.2.1.5.7 **Αποδεικνύουμε τον αλγόριθμος εξισορρόπησης φορτίου roundrobin.**

Για να αποδείξουμε ότι πραγματοποιείται η περιοδική μας εξυπηρέτηση από τους δύο κόμβους αρκεί να εκτελέσουμε το επόμενο ερώτημα στον κόμβο haproxy2 μετά την εκτέλεση του προηγούμενου πειράματος :

- SELECT VARIABLE_VALUE as "backend ID" FROM INFORMATION_SCHEMA.GLOBAL_STATUS WHERE VARIABLE_NAME="WSREP_GCOMM_UUID";

Το αποτέλεσμά του θα είναι να μας επιστρέψει το GCOMM_UUID του στιγμιότυπου της εγκατάστασης της MariaDB, που είναι σε κάθε έναν κόμβο. Οπότε θα πρέπει να δούμε τις κάτωθι εικόνες που ανήκουν στα GCOMM_UUID της mariadbcluster1 και mariadbcluster2 να εναλλάσσονται περιοδικά ανάλογα με το ποιος εξυπηρετεί το ερώτημά μας.



Εικόνα 51 : Μας εξυπηρετεί ο κόμβος mariadbcluster1

```
MariaDB [information_schema]> SELECT VARIABLE_VALUE as "backend ID" FROM INFORMATION_SCHEMA.GLOBAL_S
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    6092
Current database: information_schema

+--------------------------------------+
| backend ID                           |
+--------------------------------------+
| f9cba653-21e1-11e7-a1f8-776301994937 |
+--------------------------------------+
1 row in set (0.09 sec)
```

Εικόνα 52 : Μας εξυπηρετεί ο κόμβος mariadbcluster2

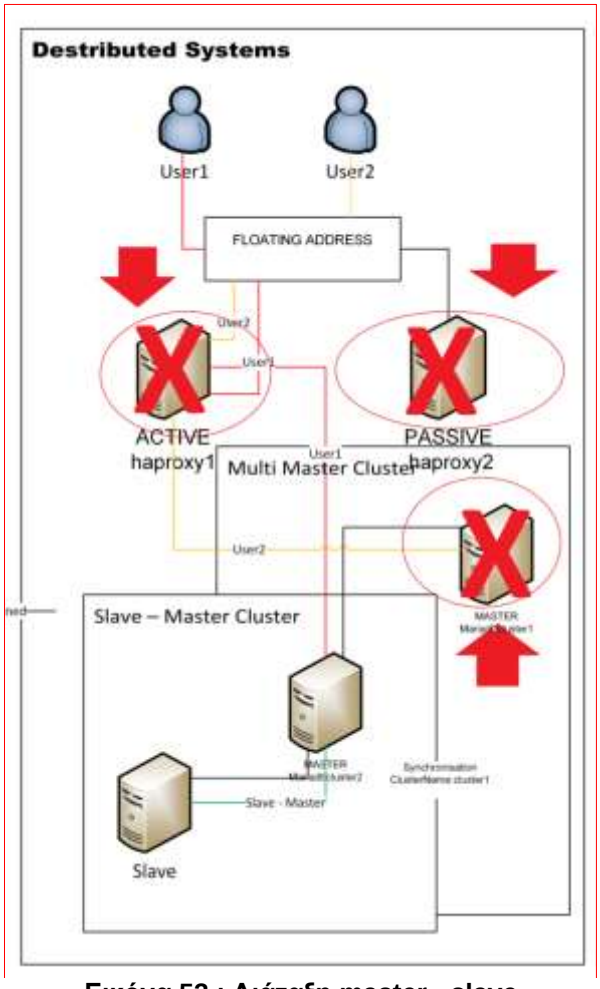
3.2.2 Υλοποίηση της διάταξης αφέντη - σκλάβου

3.2.2.1 Σκοπός

Θα δημιουργήσουμε 2 κόμβους.

Ένας θα υλοποιεί τον αφέντη και ο άλλος τον σκλάβο.

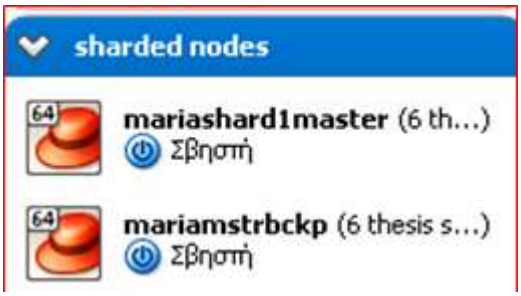
- Κόμβος1- mariashard1master: 192.168.56.231
- Κόμβος2- mariamstrbckp: 192.168.56.235



Εικόνα 53 : Διάταξη master - slave

Οι κόμβοι θα προσομοιάζουν την διάταξη αφέντη – σκλάβου, που θα υπάρχει σε κάθε ένα κέντρο δεδομένων..

3.2.2.2 Προαπαιτούμενα



Εικόνα 54:Διάταξη mariashard1master - mariamstrbckp

Σε κάθε διάταξη πολλαπλών αφεντών θα τηρούμε ένα κόμβο για τήρηση αντιγράφων ασφαλείας. Συγκεκριμένα αυτή η διάταξη θα αναφέρεται στον αφέντη κόμβο spider mariashard1master, οπότε θα τηρεί τα αποτελέσματα των συναλλαγών που θα διέρχονται από τον προηγούμενο κόμβο. Αυτά τα αποτελέσματα θα χρησιμοποιηθούν σε περίπτωση απώλειας σύνδεσης του κόμβου spider με τους κόμβους που διαθέτουν τα δεδομένα της βάσης, καθώς ο κόμβος δεν τηρεί δεδομένα παρά μόνο μια χαρτογράφηση των δεδομένων των κόμβων με τα οποία και συνδέεται. Ο κόμβος σκλάβος θα είναι ο mariamstrbckp. Δεν θα υλοποιήσουμε τους δύο haproxy καθώς και τον δεύτερο master κόμβος καθώς το κάναμε στο προηγούμενο παράδειγμα. Σε κάθε ένα vm έχουμε δώσει ένα πυρήνα, 1 GB RAM και 8 GB HDD. Το λειτουργικό που χρησιμοποιήθηκε είναι το centOs7 και συγκεκριμένα το CentOS-7-x86_64-Minimal-1611 edition.

3.2.2.3 Υλοποίηση Διάταξης

Αρχικά στον κόμβο 192.168.56.231 mariashard1master κάνουμε τις εξής αλλαγές στο αρχείο:

- /etc/my.cnf

```
[mysqld]

log-bin=mysql-bin
binlog-do-db=spider_db
server-id=1
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Εικόνα 55: Αρχείο /etc/my.cnf του mariashard1master

Αυτό έχεις σαν αποτέλεσμα να ορίσουμε ως βάση δεδομένων για να πραγματοποιήσουμε την αντιγραφή δεδομένων τον πίνακα spider_db που δημιουργήσαμε και θα δούμε ξανά στην υλοποίηση του spider.

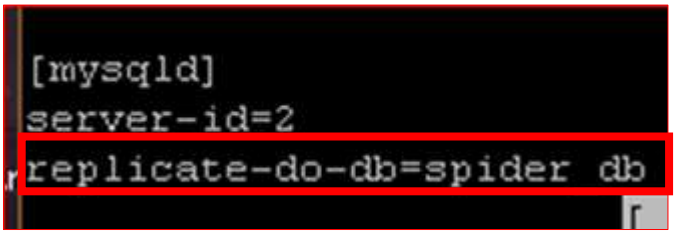
Έπειτα αναζητάμε το αρχείο από το οποίο και θα χρησιμοποιηθεί για την αντιγραφή δεδομένων στον mariamstrbckp. Οπότε όντας συνδεδεμένη στην MariaDB πραγματοποιούμε το κάτωθι ερώτημα που μας επιστρέφει τα δεδομένα που επιθυμούμε. Το αρχείο είναι το mysql-bin.000006 και ξεκινάει από την θέση 673,

```
MariaDB [(none)]> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000006 |      673 | spider_db    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Εικόνα 56:Δεδομένα master mariashard1master

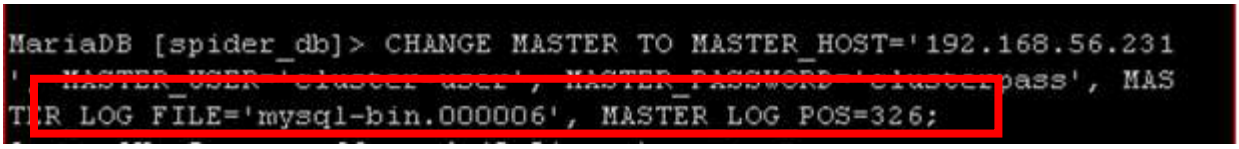
Έπειτα στον κόμβο 192.168.56.235 mariashardbackup πραγματοποιούμε τις εξής αλλαγές στον αντίστοιχο αρχείο.

- /etc/my.cnf



Εικόνα 57: Αρχείο /etc/my.cnf του mariamstrbckp.

Ακολούθως εισαγόμαστε στην MariaDB και εισάγουμε τις πληροφορίες που μας έδωσε ο mariashard1master για το αρχείο από το οποίο και θα πραγματοποιείται η καταγραφή των αλλαγών στην βάση δεδομένων μας.



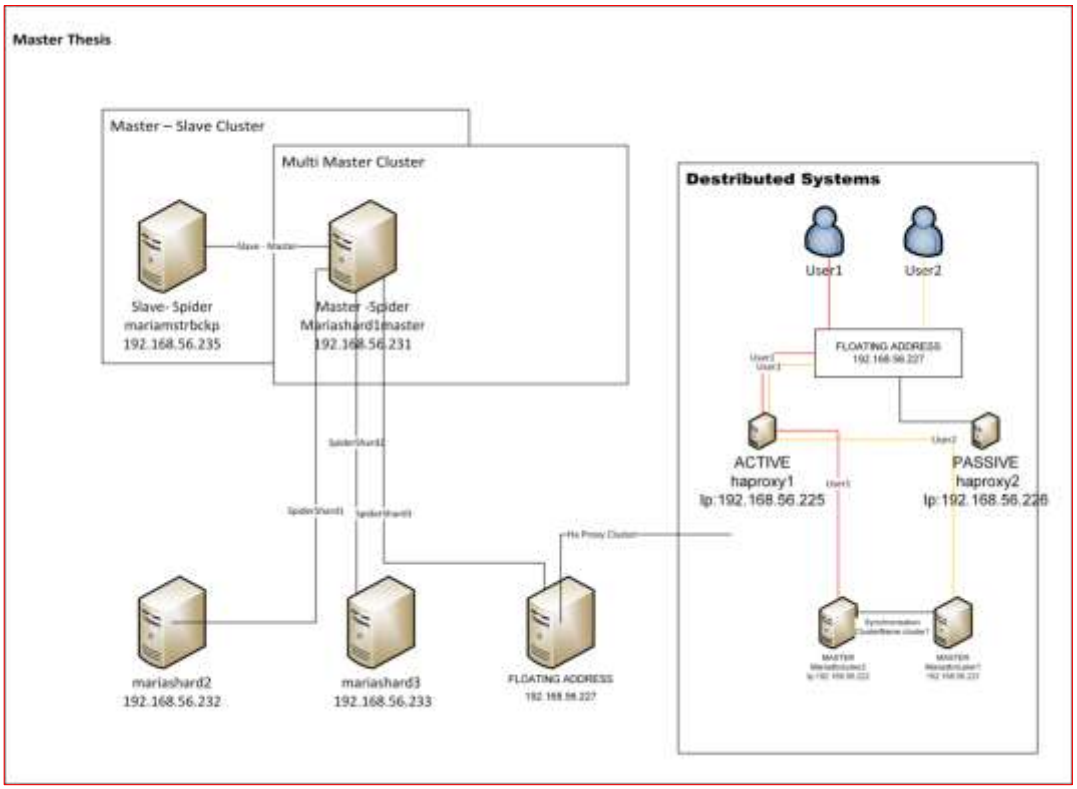
Εικόνα 58: Εισαγωγή αρχείου καταγραφής σε mariamstrbckp.

3.2.3 Υλοποίηση της διάταξης κόμβων spider.

3.2.3.1 Σκοπός

Θα δημιουργήσουμε οχτώ κόμβους .Θα υλοποιήσουμε την χαρτογράφηση των δεδομένων της βάσης μας , μέσω του κόμβου spider καθώς και την τήρηση αντιγράφων μέσω της ύπαρξης του κόμβου σκλάβου. Σε αυτό το πείραμα θα συμμετέχει το σύνολο των κόμβων που έχουμε δημιουργήσει μέχρι στιγμής:

- Κόμβος1 (spider) - mariashard1master: 192.168.56.231
- Κόμβος2 (backup) - mariamstrbckp: 192.168.56.235
- Κόμβος3 (shard1) - mariashard2: 192.168.56.232
- Κόμβος4 (shard2)- mariashard3: 1192.168.56.233
- Κόμβος5 - haproxy1: 192.168.56.225
- Κόμβος6 - haproxy2: 192.168.56.226
- Κόμβος7 (shard3)- mariadbcluster1: 192.168.56.221
- Κόμβος8 (shard3)- mariadbcluster2: 192.168.56.222
- Virtual IP Address (VIP): 192.168.56.227



Εικόνα 59 : Συνολική Διάταξη spider

3.2.3.2 Προαπαιτούμενα

Σε αυτή την διάταξη σε κάθε ένα κόμβο θα τηρείται ένα θραύσμα της βάσης (mariashard2, mariashard3, VIP). Το σύνολο των δεδομένων χαρτογραφούνται από τον mariashard1master ενώ τηρείται αντίγραφο ασφαλείας από το mariamstrbckp. Η VIP αναφέρεται στην διάταξη πολλαπλών αφεντών που δημιουργήσαμε στην αρχή της υλοποίηση των πειραμάτων και αυτή θα μας δίνει πρόσβαση στο τρίτο θραύσμα της βάσης μας. Θα χρησιμοποιηθεί το στοιχείο της υψηλής διαθεσιμότητας, της κατανομής φορτίου και των ελέγχων υγείας. Γενικά αυτή η διάταξη αποτελεί την προτεινόμενη λύση για κάθε επιχείρηση που τηρεί μια μεγάλης κεντρικοποιημένης βάση δεδομένων και επιθυμεί να την κατακερματίσει μεταξύ ενός συνόλου πολλαπλών κόμβων για να επιτευθεί κλιμακωσιμότητα και γενική βελτίωση της απόδοσης. Σε κάθε ένα vm έχουμε δώσει ένα πυρήνα, 1 GB RAM και 8 GB HDD. Το λειτουργικό που χρησιμοποιήθηκε είναι το centOs7 και συγκεκριμένα το CentOS-7-x86_64-Minimal-1611 edition.

Εικόνα 60 : Συνολικά VMs που θα χρησιμοποιηθούν στο τρίτο πείραμα

3.2.3.3 Υλοποίηση Διάταξης

Αρχικά σε κάθε ένα κόμβο θα δημιουργήσουμε την βάση spider_db και τον πίνακα sharding. Αυτό αποτελεί ένα απλό παράδειγμα για να ελέγξουμε ότι το πείραμα είναι ορθό. Στην επόμενη ενότητα θα ακολουθήσουν πιο πολύπλοκες υλοποιήσεις.

```
MariaDB [spider_db]> CREATE TABLE sharding(
    -> id INT NOT NULL,
    -> code VARCHAR(10),
    -> PRIMARY KEY(id)
    -> )ENGINE=INNODB;
Query OK, 0 rows affected (0.02 sec)
```

Εικόνα 61: Υλοποίηση πίνακα sharding

Στον κόμβο 192.168.56.231 mariashard1master που θα αποτελεί τον spider κόμβο θα ενεργοποιήσουμε την μηχανή spider μέσα από την mysql καθώς υπάρχει στην έκδοση τόσο της MariaDB όπως και της MySQL. Θα χρησιμοποιηθεί η εντολή που παρουσιάσαμε στο κεφάλαιο 2.

Έπειτα θα δημιουργήσουμε τον αντίστοιχο πίνακα spider που θα συνδέεται με τους πίνακες sharding στους αντίστοιχους κόμβους.

```
MariaDB [spider_db]> CREATE TABLE sharding(id INT NOT NULL, code VARCHAR
    -> PRIMARY KEY(id))
    -> ENGINE=SPIDER COMMENT='user "sp_user", password
    '> "1234", port "3306", table "sharding"'
    -> PARTITION BY RANGE(id)
    -> (
    -> PARTITION p1 VALUES LESS THAN (100000)
    -> COMMENT 'host "192.168.56.232"',
    -> PARTITION p2 VALUES LESS THAN (200000)
    -> COMMENT 'host "192.168.56.233"',
    -> PARTITION p3 VALUES LESS THAN MAXVALUE
    -> COMMENT 'host "192.168.56.227"'
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Εικόνα 62: Αναπαράσταση πίνακα sharding μέσω του μηχανισμού spider.

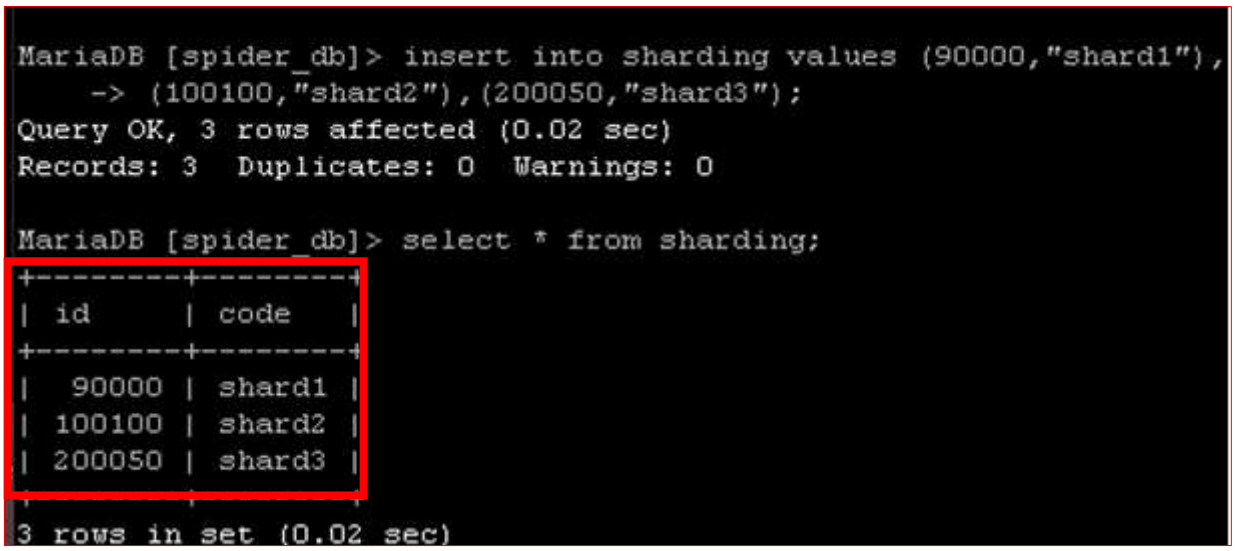
Στην παρούσα υλοποίηση έχουμε χωρίσει την βάση σε τρία τμήματα κατά εύρος.

- Όσες εγγραφές έχουν διαθέτουν τιμή id μικρότερο από 100000 ανήκουν στο τμήμα 1, όπου είναι αποθηκευμένες στον κόμβο “mariashard2: 192.168.56.232”.
- Όσες εγγραφές έχουν τιμή id μικρότερο από 200000 αλλά μεγαλύτερο από 100000 ανήκουν στο τμήμα 2, όπου είναι αποθηκευμένες στον κόμβο “mariashard3: 192.168.56.233”.
- Και τέλος οι υπόλοιπες εγγραφές ανήκουν στο τμήμα 3, όπου είναι αποθηκευμένες στην διάταξη υψηλής διαθεσιμότητας του πρώτους πειράματος, άρα είναι αποθηκευμένα στους κόμβους “mariadbcluster1: 192.168.56.221”, “mariadbcluster2: 192.168.56.222” οι οποίοι εκφράζονται από την “VIP 192.168.56.227”.

3.2.3.4 Έλεγχος επιτυχής λειτουργίας της διάταξής μας.

Για τον έλεγχο της λειτουργίας της διάταξής μας αρκεί να εισάγουμε ένα σύνολο τιμών στον κόμβο spider και να ελέγξουμε ότι κατανέμονται στους αντίστοιχους κόμβους, σύμφωνα με το εύρος που έχουμε ορίσει καθώς και το id που διαθέτουν. Επίσης ελέγχουμε ότι τηρείται ένα αντίγραφο των συναλλαγών μας από τον αντίστοιχο κόμβο.

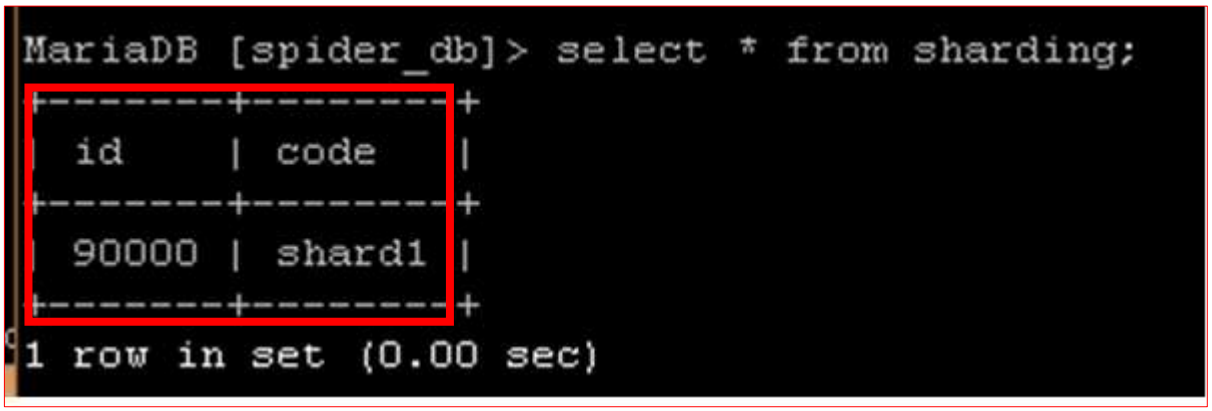
Οπότε πραγματοποιούμε είσοδο στον κόμβο 192.168.56.231 mariashard1master(spider) και εισάγουμε το παρακάτω ερώτημα:



Εικόνα 63: Εισαγωγή εγγραφών στο κόμβο spider

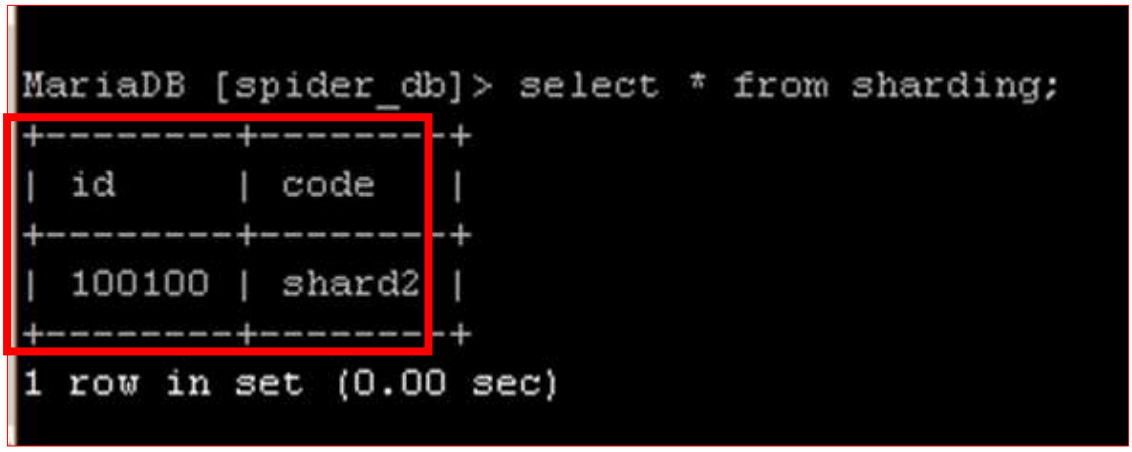
Θα πρέπει να παρατηρήσουμε στους επόμενους κόμβους την παρακάτω εικόνα:

- Στον 192.168.56.232 mariashard2 (shard1) πραγματοποιήθηκε η εισαγωγή του 90000 γιατί σύμφωνα με το εύρος που ορίσαμε αυτή η τιμή ήταν μικρότερη από 100000.



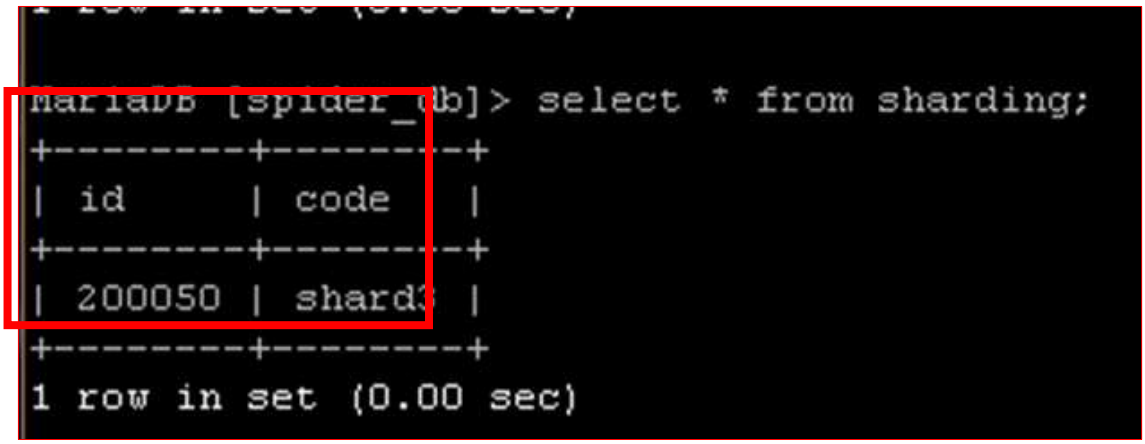
Εικόνα 64: Εγγραφές στον κόμβο 192.168.56.232 mariashard2

- Στον 192.168.56.233 mariashard3 (shard2) πραγματοποιήθηκε η εισαγωγή του 100100 γιατί σύμφωνα με το εύρος που ορίσαμε αυτή η τιμή ήταν μικρότερη από 200000 και μεγαλύτερη από 100000.



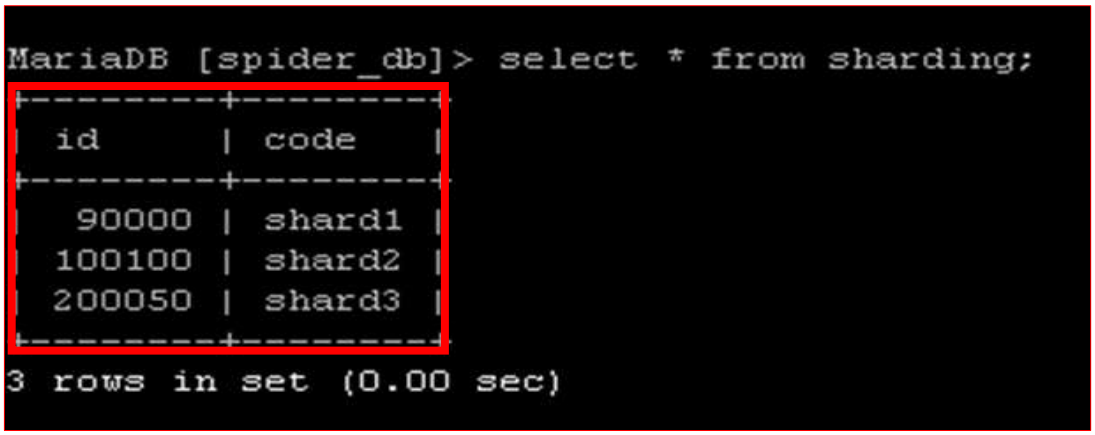
Εικόνα 65: Εγγραφές στον κόμβο 192.168.56.233 mariashard3

- Στο 192.168.56.227 VIP (shard3) πραγματοποιήθηκε η εισαγωγή του 200056 γιατί σύμφωνα με το εύρος που ορίσαμε αυτή η τιμή ήταν μεγαλύτερη από 200000.



Εικόνα 66: Εγγραφές στον κόμβο mariadbcluster1: 192.168.56.221, mariadbcluster2: 192.168.56.222

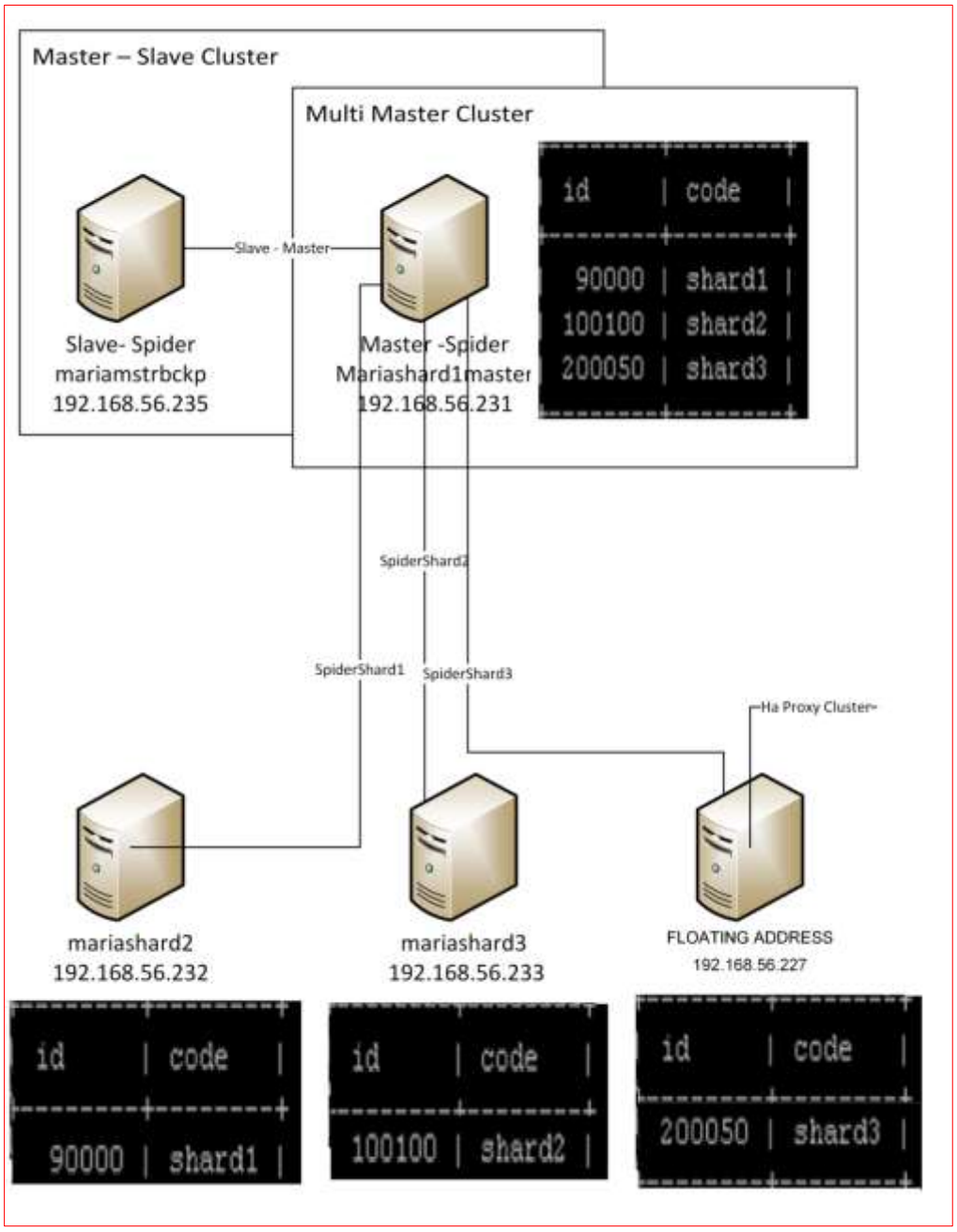
- Τέλος στον 192.168.56.235 mariashardbackup (backup) πραγματοποιήθηκε η εισαγωγή όλων των εγγραφών που πραγματοποιήθηκαν στον κόμβο spider. Επίσης αυτός ο κόμβος έχει την ιδιότητα να τηρεί πραγματικά δεδομένα της βάσης δεδομένων , σε αντίθεση με τον κόμβο spider ο οποίος αποτελεί μια χαρτογράφηση των πινάκων με τα οποία και συνδέεται.



Εικόνα 67: Εγγραφές στον κόμβο mariamstrbckp: 192.168.56.235

3.2.3.5 Γραφική αναπαράσταση της υλοποίησής μας.

Για να κάνουμε πιο κατανοητό στον αναγνώστη το πείραμα που πραγματοποιήθηκε , δημιουργήθηκε η παρακάτω γραφική αναπαράσταση.



Εικόνα 68 : Υλοποίηση

Σύμφωνα με αυτή, τα δεδομένα που εισάγονται στον πίνακα που βρίσκεται στον κόμβο spider, κατακερματίζονται μεταξύ των διαφορετικών κόμβων, σύμφωνα με το εύρος των τιμών που ορίσαμε όταν δημιουργήσαμε τον αντίστοιχο πίνακα spider.

Επίσης έχουμε φροντίσει να τηρούμε ένα αντίγραφο των συναλλαγών, πράγμα το οποίο και αποδεικνύει και το δεύτερο πείραμα που αναφερόταν στην διάταξη αφέντη – σκλάβου.

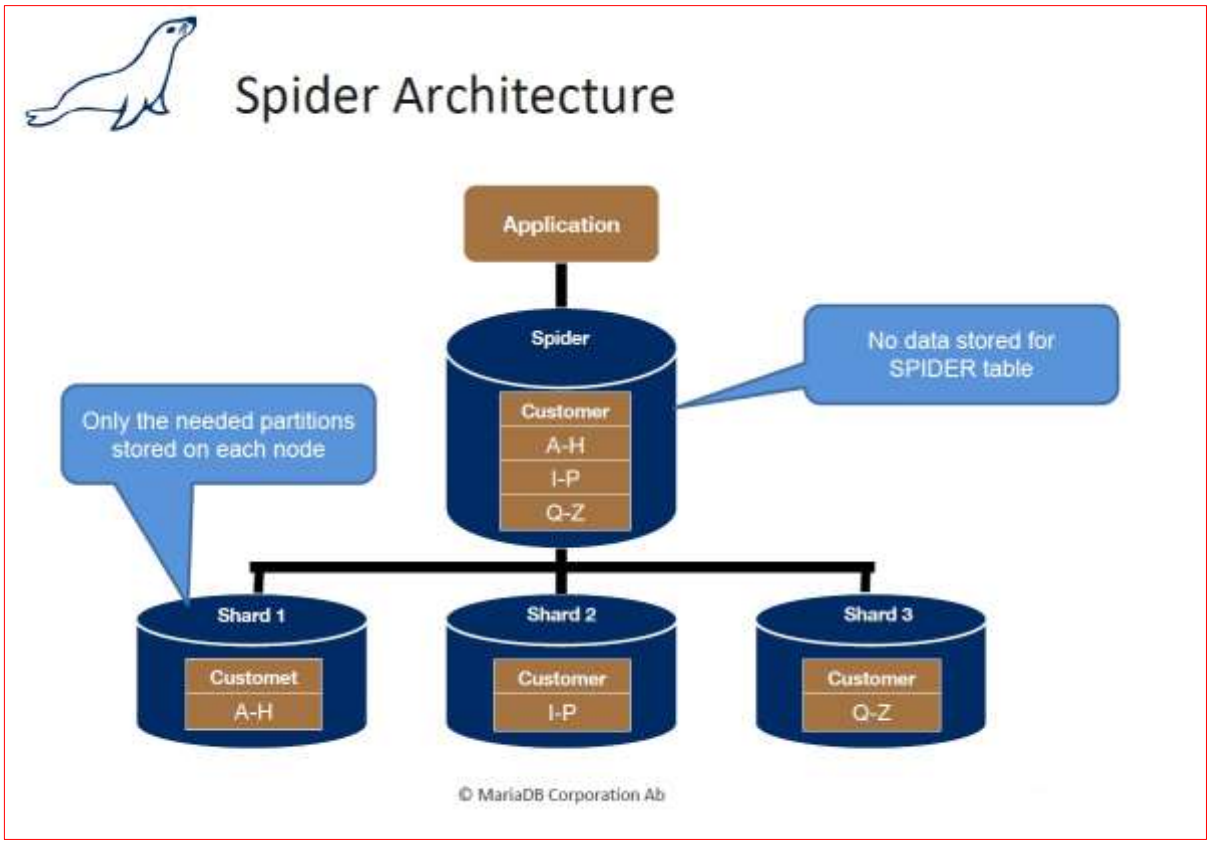
Ο αναγνώστης θα πρέπει να φανταστεί ότι κάθε ένα κόμβος που παρουσιάζεται σε αυτήν την εικόνα είναι μια συστάδα υψηλής διαθεσιμότητας εξυπηρετητών, όπως αυτή που υλοποιείται με την VIP 192.168.56.227 που τηρήσαμε στο πρώτο μας πείραμα.

Αποτέλεσμα των προηγούμενων είναι να συνδυάζουμε τα οφέλη που λάβαμε από κάθε ένα πείραμά μας επιτυγχάνοντας :

- Υψηλή διαθεσιμότητα
- Εξισορρόπηση Φορτίου
- Κατακερματισμό της βάσης
- Κλιμακωσιμότητα
- Βελτίωση της απόδοσης.
- Τήρηση αντιγράφων ασφαλείας.
- Δυνατότητα υποστήριξης μεγαλύτερου αριθμού χρηστών.
- Συνεκτικότητα της βάσης μας

Και όλα τα προηγούμενα βέβαια με ένα τρόπο διαφανή προς τον χρήστη που πραγματοποιεί σύνδεση χρησιμοποιώντας την αντίστοιχη VIP.

Η ακόλουθη εικόνα παρουσιάζει την διάταξη που θα υλοποιηθεί στο τελευταίο κεφάλαιο της μελέτης μας.



Εικόνα 69: Διάταξη spider των πειραματικών διαδικασιών.

4. ΠΕΙΡΑΜΑΤΙΚΕΣ ΜΕΤΡΗΣΕΙΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ

4.1 Εισαγωγή – Συνοπτική παρουσίαση των μετρήσεων.

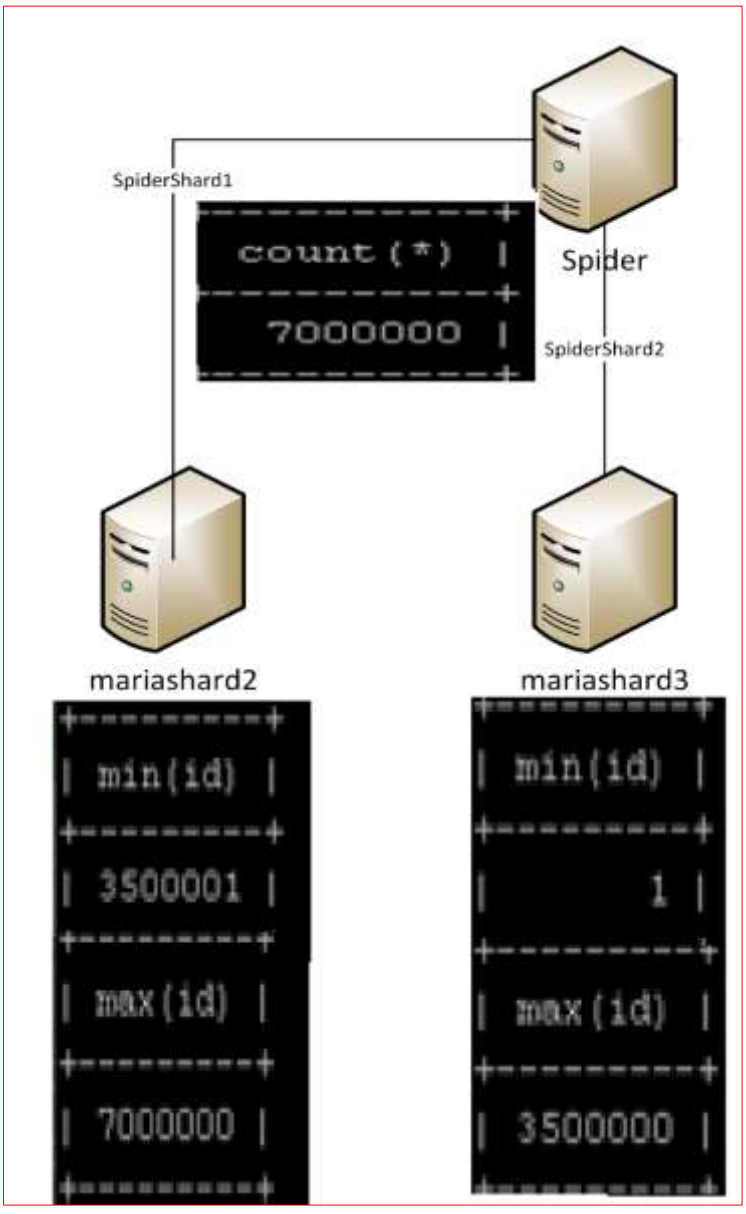
Το τελευταίο κεφάλαιο της μελέτης μας θα αναλωθεί σε μετρήσεις που πραγματοποιήθηκαν στην πειραματικής μας διάταξης. Το αποτέλεσμα αυτών των μετρήσεων θα αποδείξει την σημασία της διαδικασίας τους κατακερματισμού των βάσεων δεδομένων, χρησιμοποιώντας το μετροπρόγραμμα SysBench που παρουσιάσαμε στο κεφάλαιο 2.

4.2 Σκοπός

Θα χρησιμοποιήσουμε 3 κόμβους :

- Κόμβος1 (spider) - mariashard1master: 192.168.56.231
- Κόμβος2 (shard1) - mariashard2: 192.168.56.232
- Κόμβος3 (shard2) - mariashard3: 192.168.56.233

.



Εικόνα 70: Διάταξη που θα χρησιμοποιηθεί για την διεξαγωγή μετρήσεων

4.2.1 Προαπαιτούμενα

Σε αυτή την διάταξη κάθε ένας κόμβος στον οποίο έχει κατακερματιστεί η βάση μας (mariashard2, mariashard3) τηρεί ένα σύνολο από 3.500.000 εγγραφές σε ένα πίνακα. Αυτά χαρτογραφούνται από έναν κόμβο spider τον mariashard1master, με αποτέλεσμα να διαθέτει 7.000.000 εγγραφές στον αντίστοιχο πίνακα.

Εικόνα 71: VM που θα χρησιμοποιηθούν στη διεξαγωγή μετρήσεων.

Στην διεξαγωγή αυτή του πειράματος ένας από τους προηγούμενους κόμβους θα λάβει 7.000.000 εγγραφές, για να υλοποιηθεί η σύγκριση μεταξύ της κατακερματισμένης διάταξης του spider και της κεντρικοποιημένης διάταξης, όπου όλα τα δεδομένα τηρούνται σε έναν κόμβο.

Σε κάθε ένα vm έχουμε δώσει ένα πυρήνα, 1 GB RAM και 8 GB HDD. Κάθε ένα από αυτά τα VMs διαθέτει ένα λειτουργικό centOs7 χρησιμοποιήσαμε το CentOS-7-x86_64-Minimal-1611 edition και υποστηρίζονται από το Oracle VM VirtualBox.

4.2.2 SysBench

Στην πρώτη φάση του πειράματός μας, το μετροπρόγραμμα SysBench θα δημιουργήσει τον πίνακα sbtest1 με 7.000.000 εγγραφές στον mariashard3 και θα πραγματοποιήσουμε μετρήσεις σε ερωτήματα προς την κεντρικοποιημένη βάση .

Σε δεύτερη βάση θα εισάγουμε 3.500.000 εγγραφές στους κόμβους mariashard2 και mariashard3. Αυτό θα έχει σαν αποτέλεσμα ο mariashard1master , ο οποίος είναι ο spider κόμβος να μπορεί να χαρτογραφεί 7.000.000 εγγραφές συνολικά.

Για την διενέργεια αυτού του πειράματος δημιουργήσαμε έναν καινούργιο χρήστη που τον ονομάσαμε 'sbuser' και του δώσαμε δικαιώματα στους αντίστοιχους κόμβους.

Λεπτομέρειες για τα προηγούμενα δώσαμε στο κεφάλαιο 2 όταν και παρουσιάσαμε για πρώτη φορά το SysBench.

Μετά την επιτυχή δημιουργία των δύο πινάκων sbtest1 στους κόμβους mariashard2 και mariashard3, θα ακολουθήσει η δημιουργία του πίνακα spider στον κόμβο mariashard1master..

Σχετικά με τον μηχανισμό αποθήκευσης spider παρατηρούμε ότι:

- Οι εγγραφές που διαθέτουν τιμή μικρότερη από 3.500.000 βρίσκονται στον κόμβο “mariashard3: 1192.168.56.233”
- Οι εγγραφές που διαθέτουν τιμή μεγαλύτερη από 3.500.000 βρίσκονται στον κόμβο “mariashard2: 1192.168.56.232”.

Επίσης παρατηρώντας στην εικόνα διαπιστώνεται ότι το σύνολο εγγραφών που χαρτογραφούνται από τον πίνακα spider είναι 7.000.000 και τα οποία είναι τα συνολικά στην βάση δεδομένων μας.

```
MariaDB [backend]> CREATE TABLE backend.sbtest1
    -> (
    ->   id int(10) unsigned NOT NULL AUTO_INCREMENT,
    ->   k int(10) unsigned NOT NULL DEFAULT '0',
    ->   c char(120) NOT NULL DEFAULT '',
    ->   pad char(60) NOT NULL DEFAULT '',
    ->   PRIMARY KEY (id),
    ->   KEY k (k)
    -> )
    -> ENGINE=SPIDER COMMENT='user "sbuser", password
    '> "1234", port "3306", table "sbtest1"'
    -> PARTITION BY RANGE(id)
    -> (
    -> PARTITION p1 VALUES LESS THAN (3500000)
    -> COMMENT 'host "192.168.56.233"',
    -> PARTITION p2 VALUES LESS THAN MAXVALUE
    -> COMMENT 'host "192.168.56.232"'
    -> );
Query OK, 0 rows affected (0.00 sec)

MariaDB [backend]> select count(*) from sbtest1;
+----------+
| count(*) |
+----------+
|  7000000 |
+----------+
1 row in set (11.64 sec)
```

Εικόνα 72: Πίνακας sbtest1 του κόμβου spider

4.3 Ερωτήματα που θα χρησιμοποιηθούν προς την βάση μας.

Τα ερωτήματα που θα χρησιμοποιηθούν προς την βάση μας θα είναι τα ακόλουθα:

- Q1:select count(*) from sbtest1 where id<rows;
- Q2:select id from sbtest.sbtest1 where id< rows;
- Q3:select sum(k) from sbtest1 where id< rows;
- Q4:update sbtest1 set k=k+1 where id< rows;

Τα προηγούμενα ερωτήματα έχουν σκοπό να εξετάσουν το χρόνο απόκρισης της βάσης δεδομένων μας σε ερωτήματα τόσο αναζήτησης υπό συνθήκη, όσο και συνδυασμού αναζήτησης με εισαγωγή σε έναν ανάλογο αριθμό εγγραφών που διαθέτει η βάση μας. Τα αποτελέσματα των προηγούμενων παρουσιάζονται και αναλύονται παρακάτω.

4.3.1 Ερώτημα Q1.

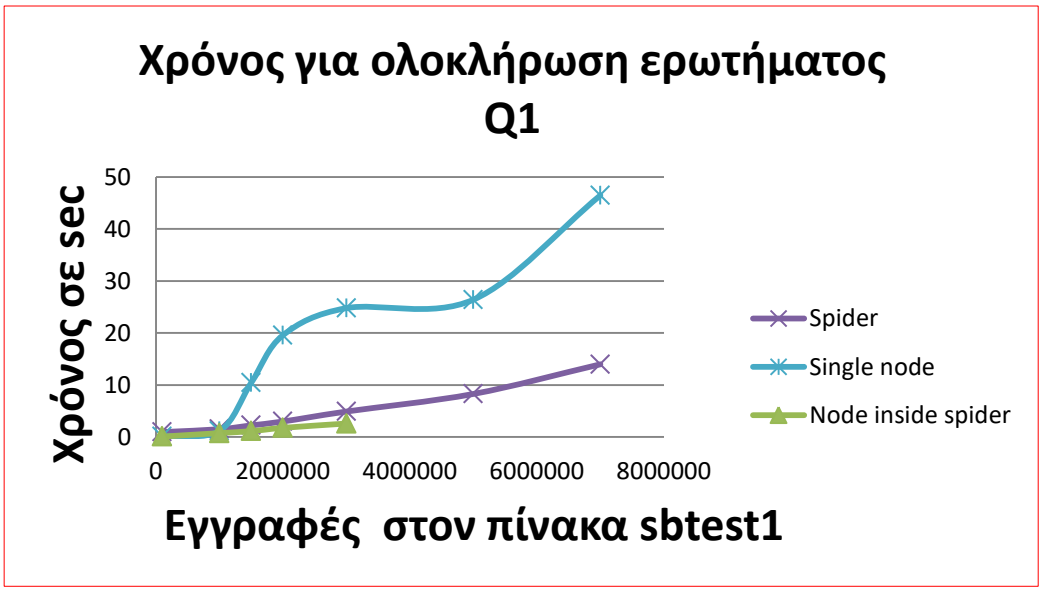
Το ερώτημα q1 επιστρέφει το σύνολο των εγγραφών, η οποία ικανοποιεί μια συνθήκη .Συγκεκριμένα αυτή η συνθήκη είναι ο αριθμός της τιμής της στήλης id να είναι μικρότερος από μια μεταβλητή που ονομάσαμε rows και παίρνει τις τιμές που περιγράφονται στον επόμενο πίνακα.

Σε αυτόν τον πίνακα εξετάζουμε τον χρόνο απόκρισης μεταξύ του κόμβου spider (1) που χαρτογραφεί 7.000.000 εγγραφές, ενός κεντρικοποιημένου κόμβου (2) που διαθέτει 7.000.000 εγγραφές και ενός κόμβου(3) που ανήκει στην διάταξη spider και διαθέτει 3.500.000 εγγραφές, όντας τμήμα μιας κατακερματισμένης βάσης δεδομένων.

Πίνακας 1: Αποτελέσματα Q1

Queries	queries	Rows	Spider (1)	Single node(2)	Node inside spider (3)
Q1	select count(*) from sbtest1 where id<100000;	100000	0,94	0,09	0,07
Q1	select count(*) from sbtest1 where id<1000000;	1000000	1,49	1,17	0,74
Q1	select count(*) from sbtest1 where id<1500000;	1500000	2,23	10,43	1,12
Q1	select count(*) from sbtest1 where id<2000000;	2000000	2,97	19,57	1,75
Q1	select count(*) from sbtest1 where id<3000000;	3000000	4,89	24,78	2,58
Q1	select count(*) from sbtest1 where id<5000000;	5000000	8,27	26,4	
Q1	select count(*) from sbtest1 where id<7000000;	7000000	13,97	46,46	

Το διάγραμμα που προκύπτει από τον προηγούμενο πίνακα είναι το επόμενο.



Σχήμα 1: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q1.

Σύμφωνα με την προηγούμενη εικόνα παρατηρούμε ότι ο κόμβος (3), ο οποίος είναι ο κόμβος ο οποίος ανήκει στην κατακερματισμένη βάση δεδομένων και διαθέτει το μισό τμήμα της βάσης μας, επιτυγχάνει να ανταποκρίνεται στο ερώτημα αναζήτησης υπό συνθήκη Q1 στο μικρότερο χρονικό διάστημα.

Σε δεύτερη θέση έρχεται ο κόμβος (1) που διαθέτει τον μηχανισμό αποθήκευσης spider.

Τέλος σε τρίτη θέση έρχεται ο κόμβος (2) που διαθέτει το σύνολο των εγγραφών όντας σε μια κεντρικοποιημένη διάταξη. Συγκεκριμένα έχοντας μεγάλη απόκλιση από τον δεύτερο και τον πρώτο.

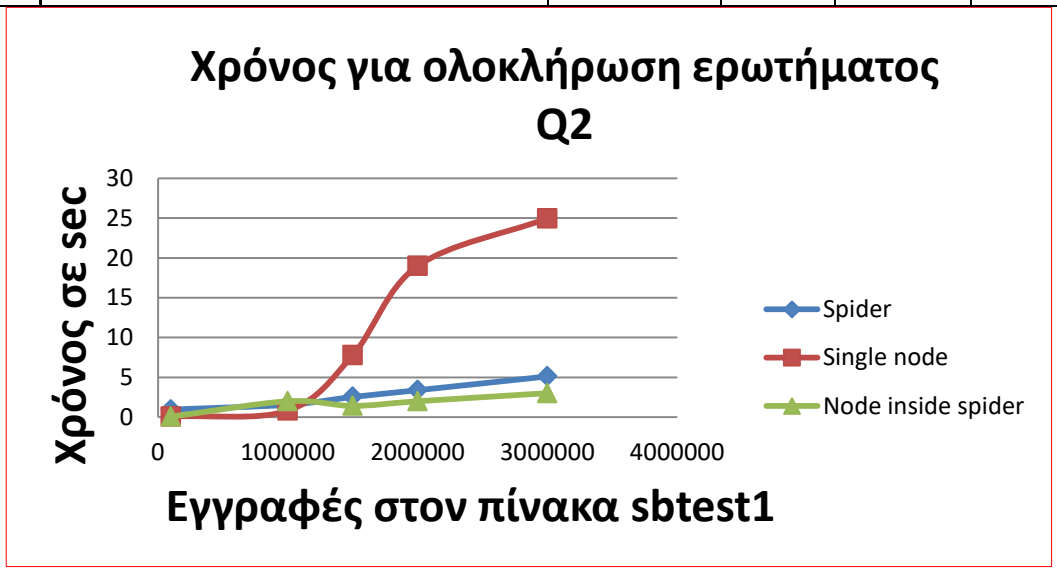
4.3.2 Ερώτημα Q2.

Το ερώτημα Q2 επιστρέφει μια λίστα με τιμές εγγραφών, η οποία ικανοποιεί μια συνθήκη .Συγκεκριμένα αυτή η συνθήκη είναι ο αριθμός της τιμής της στήλης id να είναι μικρότερος από μια μεταβλητή που ονομάσαμε rows και παίρνει τις τιμές που περιγράφονται στον επόμενο πίνακα.

Σε αυτόν τον πίνακα εξετάζουμε τον χρόνο απόκρισης μεταξύ του κόμβου spider (1) που χαρτογραφεί 7.000.000 εγγραφές, ενός κεντρικοποιημένου κόμβου (2) που διαθέτει 7.000.000 εγγραφές και ενός κόμβου(3) που ανήκει στην διάταξη spider και διαθέτει 3.500.000 εγγραφές, όντας τμήμα μιας κατακερματισμένης βάσης δεδομένων.

Πίνακας 2: Αποτελέσματα Q2

Queries	queries	Rows	Spider	Single node	Node inside spider
Q2	select id from sbtest1 where id<100000;	100000	0,96	0,08	0,07
Q2	select id from .sbtest1 where id<1000000;	1000000	1,56	0,82	2
Q2	select id from sbtest1 where id<1500000;	1500000	2,54	7,79	1,42
Q2	select id from sbtest1 where id<2000000;	2000000	3,39	18,99	2
Q2	select id from sbtest1 where id<3000000;	3000000	5,12	24,96	3,02
Q2	select id from sbtest1 where id<5000000;	5000000	10,47	25,51	
Q2	select id from sbtest1 where id<7000000;	7000000	19,69	27,46	



Σχήμα 2: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q2

Σύμφωνα με την προηγούμενη εικόνα παρατηρούμε ότι ο κόμβος (3), ο οποίος είναι ο κόμβος ο οποίος ανήκει στην κατακερματισμένη βάση δεδομένων και διαθέτει το μισό τμήμα της βάσης μας, επιτυγχάνει να ανταποκρίνεται στο ερώτημα αναζήτησης υπό συνθήκη Q2 στο μικρότερο χρονικό διάστημα.

Σε δεύτερη θέση έρχεται ο κόμβος (1) που διαθέτει τον μηχανισμό αποθήκευσης spider.

Τέλος σε τρίτη θέση έρχεται ο κόμβος (2) που διαθέτει το σύνολο των εγγραφών όντας σε μια κεντρικοποιημένη διάταξη. Συγκεκριμένα έχοντας μεγάλη απόκλιση από τον δεύτερο και τον πρώτο.

4.3.3 Ερώτημα Q3.

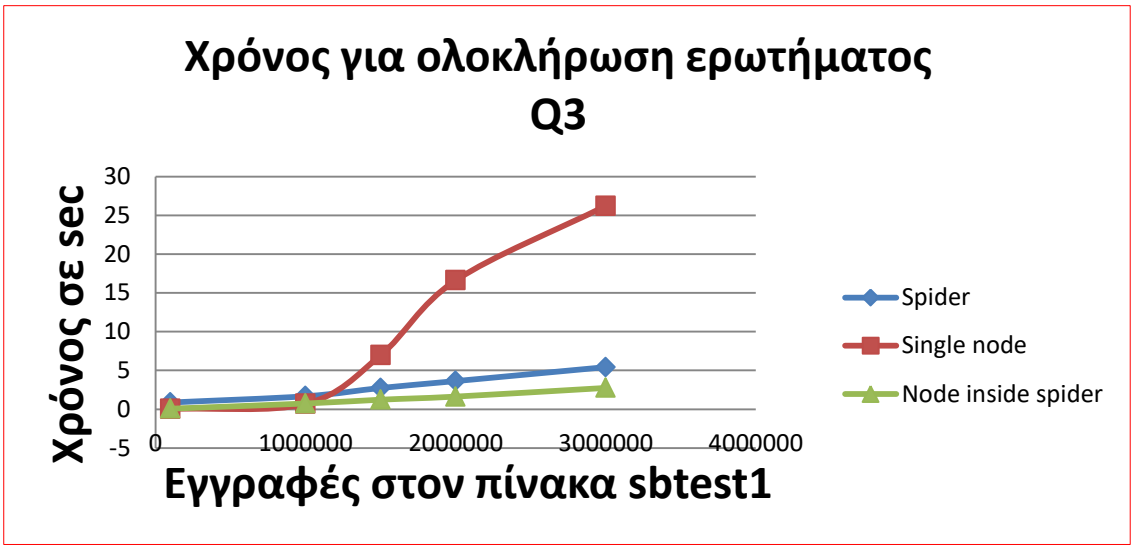
Το ερώτημα Q3 επιστρέφει το άθροισμα των τιμών της στήλης k ενός συνόλου εγγραφών όπου η τιμή της στήλης id που ικανοποιούν μια συνθήκη .Συγκεκριμένα αυτή η συνθήκη είναι ο αριθμός της τιμής της στήλης id να είναι μικρότερος από μια μεταβλητή που ονομάσαμε rows και παίρνει τις τιμές που περιγράφονται στον επόμενο πίνακα.

Σε αυτόν τον πίνακα εξετάζουμε τον χρόνο απόκρισης μεταξύ του κόμβου spider (1) που χαρτογραφεί 7.000.000 εγγραφές, ενός κεντρικοποιημένου κόμβου (2) που διαθέτει 7.000.000 εγγραφές και ενός κόμβου (3) που ανήκει στην διάταξη spider και διαθέτει 3.500.000 εγγραφές, όντας τμήμα μιας κατακερματισμένης βάσης δεδομένων.

Πίνακας 3: Αποτελέσματα Q3

Queries	queries	Rows	Spider	Single node	Node inside spider
Q3	select sum(k) from sbtest1 where id<100000;	100000	0,87	0,03	0,07
Q3	select sum(k) from sbtest1 where id<1000000;	1000000	1,68	0,69	0,74
Q3	select sum(k) from sbtest1 where id<1500000;	1500000	2,74	6,97	1,23
Q3	select sum(k) from sbtest1 where id<2000000;	2000000	3,63	16,62	1,63
Q3	select sum(k) from sbtest1 where id<3000000;	3000000	5,43	26,22	2,75
Q3	select sum(k) from sbtest1 where id<5000000;	5000000	15,05	29,99	
Q3	select sum(k) from sbtest1 where id<7000000;	7000000	23,58	38,58	

Το διάγραμμα που προκύπτει από τον προηγούμενο πίνακα είναι το επόμενο.



Σχήμα 3: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q3

Σύμφωνα με την προηγούμενη εικόνα παρατηρούμε ότι ο κόμβος (3), ο οποίος είναι ο κόμβος ο οποίος ανήκει στην κατακερματισμένη βάση δεδομένων και διαθέτει το μισό τμήμα της βάσης μας επιτυγχάνει να ανταποκρίνεται στο ερώτημα αναζήτησης υπό συνθήκη Q3 στο μικρότερο χρονικό διάστημα.

Σε δεύτερη θέση έρχεται ο κόμβος (1) που διαθέτει τον μηχανισμό αποθήκευσης spider.

Τέλος σε τρίτη θέση έρχεται ο κόμβος (2) που διαθέτει το σύνολο των εγγραφών όντας σε μια κεντρικοποιημένη διάταξη. Συγκεκριμένα έχοντας μεγάλη απόκλιση από τον δεύτερο και τον πρώτο.

4.3.4 Ερώτημα Q4.

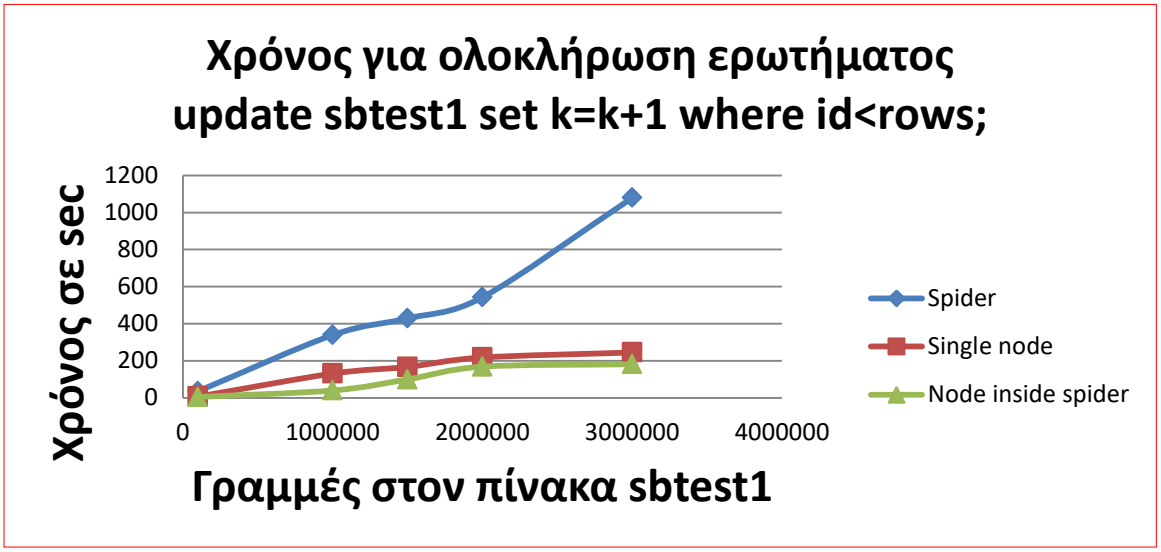
Το ερώτημα Q4 εισάγει στην στήλη k, την τιμή που περιεχόταν αυξημένη κατά ένα ψηφίο σε ένα σύνολο εγγραφών, στο οποίο η τιμή της στήλης id ικανοποιούν μια συνθήκη .Συγκεκριμένα αυτή η συνθήκη είναι ο αριθμός της τιμής της στήλης id να είναι μικρότερος από μια μεταβλητή, που ονομάσαμε rows και παίρνει τις τιμές που περιγράφονται στον επόμενο πίνακα.

Σε αυτόν τον πίνακα εξετάζουμε τον χρόνο απόκρισης μεταξύ του κόμβου spider (1) που χαρτογραφεί 7.000.000 εγγραφές, ενός κεντρικοποιημένου κόμβου (2) που διαθέτει 7.000.000 εγγραφές και ενός κόμβου (3) που ανήκει στην διάταξη spider και διαθέτει 3.500.000 εγγραφές, όντας τμήμα μιας κατακερματισμένης βάσης δεδομένων.

Πίνακας 4: Αποτελέσματα Q4

Queries	queries	Rows	Spider	Single node	Node inside spider
Q4	update sbtest1 set k=k+1 where id<100000;	100000	34,84	6,22	4,33
Q4	update sbtest1 set k=k+1 where id<1000000;	1000000	338,46	130,58	39,64
Q4	update sbtest1 set k=k+1 where id<1500000;	1500000	428,3	166,58	99,28
Q4	update sbtest1 set k=k+1 where id<2000000;	2000000	542,09	217,45	167,95
Q4	update sbtest1 set k=k+1 where id<3000000;	3000000	1078,75	244,62	181,19
Q4	update sbtest1 set k=k+1 where id<5000000;	5000000		423,82	
Q4	update sbtest1 set k=k+1 where id<7000000;	7000000		718,9	

Το διάγραμμα που προκύπτει από τον προηγούμενο πίνακα είναι το επόμενο.



Σχήμα 4: Διαγραμματική αναπαράσταση της διασποράς των τιμών του πίνακα αποτελεσμάτων Q4

Σύμφωνα με την προηγούμενη εικόνα παρατηρούμε ότι ο κόμβος (3), ο οποίος είναι ο κόμβος ο οποίος ανήκει στην κατακερματισμένη βάση δεδομένων και διαθέτει το μισό τμήμα της βάσης μας επιτυγχάνει να ανταποκρίνεται στο ερώτημα αναζήτησης υπό συνθήκης και εισαγωγής Q4 στο μικρότερο χρονικό διάστημα.

Σε δεύτερη θέση έρχεται ο κόμβος (2) που διαθέτει το σύνολο των εγγραφών όντας σε μια κεντρικοποιημένη διάταξη.

Τέλος σε τρίτη θέση έρχεται ο κόμβος (1) που διαθέτει τον μηχανισμό αποθήκευσης spider. Συγκεκριμένα έχοντας μεγάλη απόκλιση από τον δεύτερο και τον πρώτο.

4.4 Συμπεράσματα

Μετά από πειραματικές μελέτες, το συμπέρασμα που προκύπτει είναι η σημασία της ύπαρξης κατακερματισμού σε μια βάση δεδομένων.

Σε κάθε ένα ερώτημα που πραγματοποιήσαμε ο κόμβος που διέθετε την κατακερματισμένη βάση είχε καλύτερο χρόνο απόκρισης σε σχέση με έναν κεντρικοποιημένο κόμβο. Βέβαια όσο μεγαλύτερο αριθμό από κατατμήσεις της βάσης μας διαθέτουμε, τόσο πιο αποδοτικό θα είναι για τον χρόνο απόκρισης. Με αυτό τον τρόπο επιτυγχάνουμε κλιμακωσιμότητα.

Ο κόμβος ,που υποστήριζε την μηχανισμό αποθήκευσης δεδομένων spider , κατάφερε να επιφέρει σε κάθε ένα ερώτημα καλύτερη απόδοση από τον αντίστοιχο κεντρικοποιημένο κόμβο . Εξαίρεση αποτελεί μια δοκιμασία (Q4) στην οποία έπρεπε να κάνει ένα σύνολο εγγραφών στους απομακρυσμένους κόμβους και έπειτα ανάγνωσή τους. Στην προηγούμενη περίπτωση είχε το χειρότερο αποτέλεσμα. Επίσης αποδείχτηκε ότι σημασία παίζει και η καθυστέρηση του δικτύου.

Βέβαια πρέπει να αναφερθεί ότι πρωτεύων σκοπός μιας κατακερματισμένης βάσης δεδομένων έγκειται στο ότι τα δεδομένα είναι κατακερματισμένα μεταξύ ενός συνόλου κόμβων στους οποίους γίνονται ερωτήματα όπως το προηγούμενο. Οπότε δεν θα προκύψει ποτέ αυτό το πρόβλημα.

Από τις προηγούμενες παρατηρήσεις καταλήγουμε στην ορθότητα της αρχιτεκτονικής μας .Το προηγούμενο αποδείχτηκε χάρις στην δυνατότητα της να επιφέρει μικρότερο χρόνο απόκρισης σε κάθε ένα ερώτημα, σε σχέση με μια αρχιτεκτονική που διαθέτει μια κεντρικοποιημένη βάση δεδομένων.

Κατόπιν τον προηγουμένων προτείνεται αυτή η αρχιτεκτονική, καθώς όπως αποδείξαμε προσφέρει :

- Υψηλή διαθεσιμότητα
- Εξισορρόπηση φορτίου
- Κατακερματισμό της βάσης δεδομένων
- Κλιμακωσιμότητα

Και όλα τα προηγούμενα με ένα τρόπο διαφανή προς τον χρήστη ο οποίος αρκεί να διαθέτει μία διεύθυνση VIP

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
State Exchange	Ανταλλαγή Κατάστασης
Standalone	Αυτόνομους
Data Corruption	Αλλοίωση των Δεδομένων
Connection Pooling	Ανακυκλώσιμες Συνδέσεις
Tolerance to network Partitions	Ανοχή στις κατατμήσεις που οφείλονται στο δίκτυο
Replication	Αντιγραφή δεδομένων
Reverse Proxy	Αντίστροφος Μεσολαβητής
Isolation	Απομόνωση
Atomicity	Ατομικότητα
Incremental state transfer	Αύξουσα Μεταφορά Κατάστασης
Master	Αφέντης
Transactional Databases	Βάσεις δεδομένων συναλλαγών
Daemon	Δαίμονα
2 Phase Commit	Δέσμευση 2 Φάσεων
Failover	Διαδικασίες Εναλλακτικής Σύνδεσης
Availabilty	Διαθεσιμότητα
Interface	Διεπαφής
IP address	Διεύθυνση IP
Rows	Εγγραφές
Virtual Machine	Εικονική μηχανή
Virtual Internet Protocol	Εικονικό Πρωτόκολλο Ίντερνετ
IP Virtual Server	Εικονικός Εξυπηρετητής IP
Membership Control	Έλεγχος Διαχείρισης Μέλους
Health Checks	Ελέγχου Υγείας
Callbacks	Επανακλήσεις
Load Balancing	Εξισορρόπηση Φορτίου
Load Balancer	Εξισορροπιστή Φορτίου
Web server	Εξυπηρετητής δικτύου
Application server	Εξυπηρετητής εφαρμογής
Resynchronize	Επανασυγχρονιζω
Shard	Θραύσμα
Lag	Καθυστέρηση
Datacenter	Κέντρο Δεδομένων
Big Data	Μεγάλα Δεδομένα
State Snapshot Transfer	Μεταφορά Κατάστασης Στιγμιότυπου
Storage Engine	Μηχανισμός Αποθήκευσης
Durability	Μονιμότητα

Read-only	Μόνο Ανάγνωσης
Database driver	Οδηγός Βάσης
Hostname	Όνομα Διακομιστή
Sharding	Οριζόντιος Κατακερματισμός
Client	Πελάτης
Primary Component	Πρωτεύων Τμήμα
Virtual Router Redundancy Protocol	Πρωτόκολλο Εφεδρείας Εικονικού Δρομολογητή
Slave	Σκλάβος
Majority Consensus	συγκατάθεση της πλειοψηφία
Transactions	Συναλλαγές
XA transactions	Συναλλαγές XA
Stateless HTTP Connection	Σύνδεση HTTP χωρίς να τηρείται κατάσταση
Split Brain Syndrome	Σύνδρομο Διάσπασης Εγκεφάλου
Consistency	Συνέπεια
Eventual Consistency	Συνέπεια Τελικώς
Server Cluster	Συστάδα Εξυπηρετητών
Galera cluster	Συστάδα Εξυπηρετητών Γαλέρα
Database Management System	Σύστημα Διαχείρισης Βάσεων Δεδομένων.
Hardware	Υλισμικό
High Availability Proxy	Υψηλής Διαθεσιμότητας Εξυπηρετητής Μεσολάβησης
Workload	Φόρτο Εργασίας

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

ACID	Atomicity – Consistency –Isolation - Durability
CAP	Consistency-Availability-Tolerance to network Partitions
crd	Κατάσταση Υγείας - cardinality
HAProxy	High Availability Proxy
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPVS	IP Virtual Server
IST	Incremental State Transfer
JDBC	Java Database Connectivity
2PC	2 Phase Commit
PHP	Hypertext Preprocessor
SQL	Structured Query Language
SST	State Snapshot Transfer
sts	Status
TCP	Transmission Control Protocol
VIP	Virtual Internet Protocol ή Floating IP
VM	Virtual Machine
VRRP	Virtual Router Redundancy Protocol
xinetd	Extended Internet Daemon
ΚΜΕ	Κεντρική Μονάδα Επεξεργασίας
ΒΔ	Βάση Δεδομένων
ΣΔΒΔ	Σύστημα Διαχείρισης Βάσεων Δεδομένων

ΑΝΑΦΟΡΕΣ

Τα url έχουν περικοπεί με το https://goo.gl/ για λόγο ομοιομορφίας.

[1] CNN.com article: iPhone snobbery greets Instagram's Android app, https://goo.gl/74Jydi [Προσπελάστηκε 22/8/17]
[2] Pinterest: Sharding Pinterest: How we scaled our MySQL fleet, https://goo.gl/4WrS5B [Προσπελάστηκε 22/8/17]
[3] Facebook: How sharding works, https://goo.gl/yZUZ2J [Προσπελάστηκε 22/8/17]
[4] Twiter: How sharding works, https://goo.gl/xLcsTq [Προσπελάστηκε 22/8/17]
[5] Esen Sagyrov "Easy Mysql Database Sharding with CUBRID SHARD" (MWCE 13) p.4 https://goo.gl/enXqKr [Προσπελάστηκε 22/8/17]
[6] Δημήτρης Κακλαμάνης (ΠΑΜΑ) , "Distributed Applications and E Commerce" , https://goo.gl/rieJxA [Προσπελάστηκε 22/8/17]
[7] MariaDB knowledge base: Synchronous vs. Asynchronous Replication, https://goo.gl/7iz2zf [Προσπελάστηκε 22/8/17]
[8] Reuter, Andreas; Haerder, Theo (December 1983). «Principles of Transaction-Oriented Database Recovery» (PDF). ACM Computing Surveys 15 (4): 287–317, https://goo.gl/4Aqj7z [Προσπελάστηκε 25/8/17]
[9] Wikipedia: What is ACID? https://goo.gl/A3DzoP [Προσπελάστηκε 25/8/17]
[10] Boris Hristov: Database Transactions and SQL Server Concurrency, https://goo.gl/m4jPsG [Προσπελάστηκε 25/8/17]
[11] University of Babylon: C. J. Date's twelve commandments for distributed databases, https://goo.gl/OcFpYN [Προσπελάστηκε 23/8/17]
[12] E.A.Brewer, (2000, Jul)Towards robust distributed systems (Invited Talk) Principle of Distributed Computing, Portland, Oregon , July 2000.
[13] E.A.Brewer, (2000, Jul)Towards robust distributed systems (Notes), https://goo.gl/TAIt5 [Προσπελάστηκε 23/8/17]
[14] N.Lynch, S.Gilbert,(2002, Nov)Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, https://goo.gl/QNJUW1 [Προσπελάστηκε 23/8/17]
[15] MariaDB knowledge base: What is MariaDB Galera Cluster? https://goo.gl/EJUz8U [Προσπελάστηκε 23/8/17]
[16] MariaDB knowledge base: Wsrep API?, https://goo.gl/8Vf21 [Προσπελάστηκε 23/8/17]
[17] Galera Cluster Documentation: How Galera Cluster Works? https://goo.gl/1eeD67 [Προσπελάστηκε 23/8/17]
[18] Galera cluster: Split Brain, https://goo.gl/LJTMmZ [Προσπελάστηκε 23/8/17]
[19] Galera Cluster Documentation: Restarting the cluster, https://goo.gl/DysWy8 [Προσπελάστηκε 23/8/17]
[20] Severalnines.com theory: Mysql Load Balancing Harpoxy Tutorial, https://goo.gl/ZX11nN [Προσπελάστηκε 23/8/17]
[21] Linux Virtual Server: What is IPVS ? https://goo.gl/374Ha [Προσπελάστηκε 24/8/17]
[22] Oracle® Linux Administrator's Guide for Release 6: What is Keepalived? https://goo.gl/Wm55qd [Προσπελάστηκε 24/8/17]
[23] Max Mether :Sharding your data with Spider Scale 13th Annual Southern California Linux Expo: https://goo.gl/A9LEne [Προσπελάστηκε 24/8/17]
[24] MariaDB: Spider Storage Engine Overview, https://goo.gl/G1KKUt [Προσπελάστηκε 24/8/17]
[25] MariaDB: XA Transactions, https://goo.gl/kHS96t [Προσπελάστηκε 24/8/17]
[26] MariaDB: Spider Storage Engine Core Concepts, https://goo.gl/GAZPMY [Προσπελάστηκε 24/8/17]
[27] MariaDB: Spider Use Cases, https://goo.gl/xDngv4 [Προσπελάστηκε 24/8/17]
[28] Frédéric Descamps: "MySQL Partitioning: The Spider Solution" Persona Live London 2011 (Notes) ,https://goo.gl/TTaFAz [Προσπελάστηκε 24/8/17]
[29] Alexey Kopytov SysBench manual, https://goo.gl/kHCnpc [Προσπελάστηκε 24/8/17]
[30] Percona repository for sysbench, https://goo.gl/BL5VoY . [Προσπελάστηκε 24/8/17]
[31] MariaDB: How to setup up mariadb with virtual box, https://goo.gl/AZJguU . [Προσπελάστηκε 25/8/17]
[32] Galera Cluster: Starting the cluster, https://goo.gl/U3AfNM [Προσπελάστηκε 25/8/17]