

Ατομική Διπλωματική Εργασία

**Υλοποίηση και Αποτίμηση μιας
Blockchain Βάσης για IoT Δεδομένα**

Ηρόδοτος Δημητρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιανουάριος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση και Αποτίμηση μιας Blockchain Βάσης για IoT Δεδομένα

Ηρόδοτος Δημητρίου

Επιβλέπων Καθηγητής

Δημήτρης Ζεϊναλιπούρ

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των
απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του
Πανεπιστημίου Κύπρου

Ιανουάριος 2021

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, δεν θα μπορούσα να μην ευχαριστήσω όσα άτομα συνέβαλαν στην επίτευξη του στόχου μου.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Δημήτρη Ζεϊναλιπούρ του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, για την καθοδήγηση και την διαρκή στήριξη που μου προσέφερε καθ' όλη την διάρκεια του περασμένου χρόνου. Η εκπαίδευση μας υπό τις πρωτοφανείς συνθήκες που ζήσαμε λόγω της πανδημίας ήταν πολύ διαφορετική από ότι συνηθίσαμε και είχε τις δυσκολίες της. Όμως μέσα πολλές διαδικτυακές συναντήσεις που είχα μαζί του, οι στόχοι που θέσαμε επιτεύχθηκαν. Ακόμη, θα ήθελα να τον ευχαριστήσω για όλες τις συμβουλές και υποδείξεις του, καθώς επίσης και για τις γνώσεις και δεξιότητες που απέκτησα συζητώντας μαζί του στα πλαίσια αυτής της πτυχιακής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια μου, η οποία με στήριξε οικονομικά και ψυχολογικά στην όλη προσπάθεια μου καθ' όλη την διάρκεια των σπουδών μου. Ιδιαίτερα, θα ήθελα να ευχαριστήσω τον πατέρα μου Δημήτρη και την μητέρα μου Μαρία, οι οποίοι ήταν πάντα στο πλευρό μου και ειδικότερα τον τελευταίο χρόνο όπου οι αντοχές όλων μας δοκιμάστηκαν μέσα από την πανδημία του κορωνοϊού.

Τέλος θα ήθελα να ευχαριστήσω την προπτυχιακή φοιτήτρια Σταυρούλα Κούμου και τον διδακτορικό φοιτητή Πάνο Δρακάτο για την συμβολή τους στην έρευνα που διεξάχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας.

Περίληψη

Τα τελευταία χρόνια, η αναδυόμενη και πολλά υποσχόμενη τεχνολογία αλυσίδας συστοιχιών (*Blockchain*) έχει κεντρίσει το ενδιαφέρον της επιστημονικής κοινότητας που ασχολείται με τις Βάσεις Δεδομένων. Εν τούτης, οι υφιστάμενες λύσεις που αφορούν την αποθήκευση, επεξεργασία και διαμοιρασμό δεδομένων, πάσχουν από διάφορα προβλήματα και χαρακτηρίζονται από μεγάλα κόστη συντήρησης ενώ προσφέρουν σχετικά χαμηλές επιδόσεις. Πέραν τούτου, ο κεντριοποιημένος χαρακτήρας διαχείρισης δεδομένων με συμβατικές τεχνολογίες, φέρει έντονες ανησυχίες όσον αφορά την ασφάλεια και διαφάνεια των συστημάτων αυτών. Το γεγονός αυτό, εγείρει μεγάλο προβληματισμό για το πως μπορούμε να βελτιώσουμε την διαχείριση δεδομένων αντικειμένων διαδικτύου (*IoT*).

Σε αυτή την διπλωματική εργασία αποφασίσαμε να παρεκλείνουμε από τις υπάρχουσες προσεγγίσεις και να αναζητήσουμε λύσεις για τα προαναφερθέντα προβλήματα μέσα από την πρόταση της αρχιτεκτονικής *TriaBase*. Η *TriaBase* αποτελεί μια ιδιωτική αλυσίδα συστοιχιών με δικαιώματα (*Private Permissioned Blockchain*), που αφορά βιομηχανικά *IoT* δεδομένα, η οποία στηρίζεται στο εργαλείο *Hyperledger Fabric*. Ειδικότερα, προτείνουμε ένα σύστημα *Blockchain* το οποίο στηρίζεται σε τεχνικές μηχανικής μάθησης στα άκρα (*Edge*), με απώτερο σκοπό να αυξήσουμε την απόδοση του συστήματος. Αυτό επιτυγχάνεται μειώνοντας την άσκοπη μεταφορά όλων των δεδομένων που παράγουν οι αισθητήρες, σε ένα δίκτυο *Blockchain*. Αντ' αυτού, τα δεδομένα αυτά χρησιμοποιούνται για την εκπαίδευση νευρωνικών μοντέλων τα οποία σε μεταγενέστερο στάδιο διαδίδονται σε όλους τους ομότιμους κόμβους της *TriaBase*.

Στην αρχιτεκτονική *TriaBase* υπάρχουν τρία κυρίως επίπεδα. Το επίπεδο άκρων (*Edge Layer*), όπου γεννιούνται τα δεδομένα, το επίπεδο εφαρμογής (*Application Layer*) όπου επεξεργάζονται τα δεδομένα και το επίπεδο δικτύου αλυσίδας συστοιχιών (*Blockchain*) όπου αποθηκεύονται και διανέμονται τα δεδομένα. Αναλυτικότερα, όπως προαναφέρθηκε τα μοντέλα εκπαιδεύονται με δεδομένα που παράγουν οι *IoT* συσκευές. Στο επίπεδο εφαρμογής δίνεται η δυνατότητα σε κάποιον εξουσιοδοτημένο χρήστη, να προβεί στην διαδικασία προώθησης των δεδομένων στο υπόλοιπο δίκτυο για αποθήκευση, ενώ ταυτόχρονα σε αυτό το σημείο μπορεί να υποβάλει επερωτήσεις στο τοπικό αντίγραφο του καθολικού (*Ledger*). Το τρίτο και τελευταίο επίπεδο ασχολείται με την διανομή και

αποθήκευση των δεδομένων σε ένα κατανεμημένο σύστημα αλυσίδας συστοιχιών (*Blockchain*) το οποίο χτίστηκε με την χρήση του εργαλείου *Hyperledger Fabric*.

Για την αποτίμηση της *TriaBase* έγιναν πειράματα όσον αφορά την εισαγωγή δεδομένων ως επίσης την υποβολή επερωτήσεων. Επίσης, στα πειράματα που έγιναν πραγματοποιήθηκε σύγκριση της συμβατικής μεθόδου υποβολής επερωτήσεων στο σύστημα, όπου κάθε δοσοληψία (*transaction*) αντιπροσωπεύει μία εγγραφή, σε σχέση με την δική μας προσέγγιση όπου μια δοσοληψία (*transaction*) σχετίζεται με ένα μοντέλο μηχανικής μάθησης.

Τα αποτελέσματα ως προς την υποβολή επερωτήσεων ήταν πολύ θετικά, αφού με την χρήση μοντέλων μηχανικής μάθησης μπορούμε να έχουμε σαφώς καλύτερες επιδόσεις. Όσον αφορά την εισαγωγή δεδομένων τα αποτελέσματα δεν είναι ενθαρρυντικά, αλλά αυτό ήταν αναμενόμενο, αφού το μεγάλο μέγεθος ενός μοντέλου μηχανικής μάθησης φέρει συμφόρηση στην διανομή και αποθήκευσή του στο κατανεμημένο δίκτυο.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή	1
1.1 Παρακίνηση.....	1
1.2 Πρόβλημα	2
1.3 Συνεισφορά.....	3
1.4 Περίγραμμα Εργασίας.....	6
Κεφάλαιο 2 Σχετική Δουλειά	9
2.1 Είδη Αλυσίδων Συστοιχιών	10
2.1.1 Δημόσιες	10
2.1.2 Ιδιωτικές.....	11
2.1.3 Με Παροχή Άδειας	11
2.1.4 Χωρίς Χρήση Άδειας.....	12
2.2 Ερευνητικές Τεχνολογίες Αλυσίδων Συστοιχιών.....	12
2.2.1 AnyLog: A Grand Unification of the Internet of Things.....	12
2.2.2 CAPER: A Cross-Application Permissioned <i>Blockchain</i>	15
2.3 <i>Influxdb</i>: Μια Βάση Δεδομένων για Χρονοσειρές	20
2.3.1 Βασικές Έννοιες	20
2.3.2 Εσωτερικοί Μηχανισμοί της Βάσης	21
2.4 Διαχείριση Δεδομένων Διαδικτύου Αντικειμένων με το Ενδιάμεσο Λογισμικό Node-RED.....	22
2.5 Αποσύνθεση Μεγάλων Δεδομένων	23
2.6 Τεχνική Ομοσπονδιακής Μάθησης.....	26
Κεφάλαιο 3 Τεχνολογικό Υπόβαθρο.....	28
3.1 Δίκτυα Ομότιμων Κόμβων.....	29
3.2 Κρυπτογραφία.....	29
3.2.1 Κρυπτογραφία Δημόσιου Κλειδιού	30
3.2.2 Συναρτήσεις Κατακερματισμού	30
3.2.3 Ψηφιακές Υπογραφές	31

3.3 Κωδικοποίηση Base64.....	32
3.4 Hyperledger Fabric.....	32
3.3.1 Δοσοληψίες.....	34
3.3.2 Το μοντέλο του Hyperledger Fabric	34
3.3.3 Αλυσίδα Συστοιχιών	36
3.3.4 Ομότιμοι Κόμβοι	37
3.3.5 Υπηρεσία Σειριοποίησης	39
3.3.6 Έξυπνα Συμβόλαια	41
3.3.7 Καθολικό.....	42
Κεφάλαιο 4 Αρχιτεκτονική TriaBase.....	44
4.1 Η ανατομία της TriaBase.....	46
4.1.1 Τμήμα Χρήστη.....	47
4.1.1.1 Διεργασίες Τμήματος Χρήστη.....	47
4.1.1.2 Διαδικασία Ομοσπονδιακής Μάθησης	49
4.1.2 Τμήμα Αλυσίδας Συστοιχιών	50
4.1.2.1 Έξυπνο Συμβόλαιο RAT	51
4.1.2.3 Έξυπνο Συμβόλαιο MAT.....	54
Κεφάλαιο 5 Πειραματική Μεθοδολογία	56
5.1 Δίκτυο Δομικών.....	56
5.2 Δεδομένα	57
5.3 Πειραματική διαδικασία	59
Κεφάλαιο 6 Πειραματική Αποτίμηση.....	63
6.1 Προσέγγιση RAT	64
6.1.1 Εισαγωγή δεδομένων	64
6.1.2 Υποβολή ερωτήσεων	65
6.2 Προσέγγιση MAT.....	66
6.2.1 Εισαγωγή δεδομένων	66
6.2.2 Υποβολή ερωτήσεων	68
6.3 Σύγκριση RAT και MAT	68

Κεφάλαιο 7 Συμπεράσματα	71
7.1 Συμπεράσματα	71
7.2 Μελλοντική δουλειά	73
Αναφορές	75
Παράρτημα «Α».....	A-1
Παράρτημα «Β».....	B-1

Κατάλογος Σχημάτων

Σχήμα 1.1 Αρχιτεκτονική <i>TriaBase</i>	4
Σχήμα 2.1 Διαδικασία κατασκευής query	13
Σχήμα 2.2 AnyLog Execution Flow	14
Σχήμα 2.3 Παράλληλα <i>Blockchain</i> με την χρήση DAG.....	16
Σχήμα 2.4 Global Consensus using <i>Orderers</i>	17
Σχήμα 2.5 Hierarchical Global Consensus	18
Σχήμα 2.6 One level Global Consensus.....	19
Σχήμα 2.7 Παράδειγμα <i>Point</i> στην <i>Influxdb</i>	21
Σχήμα 2.8 Παράδειγμα αρχείου TSM	22
Σχήμα 2.9 Industrial IoT Architecture Example	23
Σχήμα 2.10 Αποτίμηση TBD-DP σε σχέση με τον χώρο και την ακρίβεια προβλέψεων ..	25
Σχήμα 2.11 Data Postdiction	25
Σχήμα 2.12 Τα εννοιολογικά βήματα για τον αλγόριθμο κατασκευής και ανάκτησης.....	26
Σχήμα 2.13 Το μοντέλο του <i>Federated Learning</i>	27
Σχήμα 3.1 Παράδειγμα Caesar Cipher	29
Σχήμα 3.2 Παράδειγμα χρήσης Public Key Cryptography	30
Σχήμα 3.3 Διαδικασία Digital Signing	31
Σχήμα 3.4 Διαδικασία Digital Signature Verification	32
Σχήμα 3.5 <i>Hyperledger Fabric</i> Transaction Flow	35
Σχήμα 3.6 Παράδειγμα <i>Blockchain</i> με 3 Block	37
Σχήμα 3.7 Παράδειγμα Δικτύου	38
Σχήμα 3.8 Αλληλεπίδραση εφαρμογής Peer	39
Σχήμα 3.9 Παράδειγμα γεμάτου και μισογεμάτου Block.....	40
Σχήμα 3.10 Παράδειγμα διανομής Block από τον <i>Orderer</i> στα <i>Peers</i>	41
Σχήμα 3.11 Αναπαράσταση του <i>Ledger</i> στο <i>Hyperledger Fabric</i>	43
Σχήμα 4.1 Αρχιτεκτονική <i>TriaBase</i>	46
Σχήμα 4.2 Δομή κόμβου <i>TriaBase</i>	47
Σχήμα 4.3 Παράδειγμα <i>Transaction</i> για την προσέγγιση RAT.....	51
Σχήμα 4.4 Παράδειγμα <i>Transaction</i> για την προσέγγιση MT.....	54
Σχήμα 5.1 Docker Report after executing the command " docker ps "	57
Σχήμα 5.2 Benchmarking Tool.....	60

Σχήμα 6.1 Αναπαράσταση <i>Throughput</i> και <i>Latency</i> για την εισαγωγή δεδομένων στην προσέγγιση RAT.....	64
Σχήμα 6.2 Αναπαράσταση <i>Throughput</i> και <i>Latency</i> για την υποβολή επερωτήσεων στην προσέγγιση RAT.....	65
Σχήμα 6.3 Αναπαράσταση <i>Throughput</i> και <i>Latency</i> για την εισαγωγή δεδομένων στην προσέγγιση MAT.....	66
Σχήμα 6.4 Αναπαράσταση <i>Throughput</i> και <i>Latency</i> για την υποβολή επερωτήσεων στην προσέγγιση MAT.....	68
Σχήμα 6.5 Σύγκριση RAT και PAT όσον αφορά το <i>Throughput</i> εισαγωγής δεδομένων...	69
Σχήμα 6.6 Σύγκριση RAT και PAT όσον αφορά το <i>Throughput</i> υποβολής επερωτήσεων	70

Κατάλογος Πινάκων

Πίνακας 2.1 Κατηγορίες <i>Blockchain</i>	10
Πίνακας 4.1 Εισαγωγή δεδομένων	48
Πίνακας 4.2 Υποβολή επερωτήσεων	49
Πίνακας 4.3 Διαδικασία <i>Federated Learning</i>	50
Πίνακας 5.1 <i>Telco</i> Record Format	58
Πίνακας 5.2 <i>Machine learning</i> Model Format	59

Κεφάλαιο 1

Εισαγωγή

1.1 Παρακίνηση.....	1
1.2 Πρόβλημα	2
1.3 Συνεισφορά.....	3
1.4 Περίγραμμα Εργασίας.....	6

1.1 Παρακίνηση

Στην εποχή την οποία ζούμε, η τεχνολογική πρόοδος έχει θέσει νέα πρότυπα στην καθημερινότητά μας. Ονομαστικά, η έλευση νέων τεχνολογιών όπως το 5G και το διαδίκτυο αντικειμένων (Internet of Things), δημιουργούν νέες ανάγκες για επεξεργασία και αποθήκευση μεγάλου όγκου δεδομένων, που παράγουν οι έξυπνες συσκευές. Το γεγονός αυτό εγείρει μια πληθώρα από ερωτήματα και προκλήσεις. Σαφώς, οι ευκαιρίες για έρευνα στον τομέα του *IoT* είναι πολλές και συμπεριλαμβάνουν ένα ευρύ φάσμα από εφαρμογές, όπως το μηχανική μάθηση (*machine learning*), η επαυξημένη πραγματικότητα (*augmented reality*), η παρακολούθηση και ο έλεγχος έξυπνων σπιτιών (*Smart Homes*) κλπ.

Σε κάθε σενάριο υλοποίησης μιας *IoT* αρχιτεκτονικής, η χρήση βάσης δεδομένων για την αποθήκευση πληροφοριών είναι απαραίτητη. Ειδικότερα, μια βάση δεδομένων βρίσκεται στο πίσω μέρος (*back-end*) της υποδομής και εξυπηρετεί πολλαπλά μέλη που συμμετέχουν σε ένα σύμπλεγμα από *IoT* συσκευές. Συνεπώς, μια κοινόχρηστη βάση δεδομένων γεφυρώνει το χάσμα μεταξύ του επίπεδου γένεσης των δεδομένων στα άκρα (*Edge*) και του επίπεδου ανάλυσης των δεδομένων σε πραγματικό χρόνο.

Πέραν τούτου η χρήση *Machine learning* σε διάφορους τομείς της καθημερινότητάς μας, είναι συχνό φαινόμενο. Πλέον, όλοι οι επεξεργαστές που κυκλοφορούν στην αγορά είτε αυτοί αφορούν κινητές συσκευές, είτε αφορούν προσωπικούς υπολογιστές είτε διακομιστές (*Servers*), διαθέτουν μονάδα επεξεργασίας μοντέλων *Machine learning*. Πιο αναλυτικά, η χρήση *Machine learning* μπορεί να λύσει πολλά προβλήματα, όπως την αναγνώριση και πρόβλεψη μοτίβων σε δεδομένα, και να βελτιστοποιήσει την επεξεργασία και την αποθήκευσή τους.

Παράλληλα, η αναδυόμενη και πολλά υποσχόμενη τεχνολογία αλυσίδας συστοιχιών (*Blockchain*), δύναται για την εφαρμογή της σε διάφορους τομείς συμπεριλαμβανομένων του τραπεζικού τομέα, αποκεντροποίησης του διαδικτύου και του Internet of Things. Το γεγονός αυτό μας κίνησε το ενδιαφέρον και ως εκ τούτου στα πλαίσια της παρούσας διπλωματικής εργασίας αποφασίσαμε να ερευνήσουμε το σενάριο χρήσης της τεχνολογίας *Blockchain* για την διαχείριση *IoT Machine learning* μοντέλων.

1.2 Πρόβλημα

Μερικά από τα πιο κοινά προβλήματα που ταλανίζουν τα αποκεντρωμένα συστήματα, αφορούν την ασφαλή μεταφορά δεδομένων από τα άκρα σε όλο το δίκτυο ως επίσης και την κλιμακωσιμότητα ενός δικτύου, όταν ο ρυθμός δεδομένων που εισάγεται σε αυτό αυξάνεται.

Αρχικά, οι παραδοσιακές υποδομές που χρησιμοποιούνται για την αποθήκευση και επεξεργασία *IoT* δεδομένων έχουν πεπερασμένο χώρο αποθήκευσης, ενώ ταυτόχρονα οι περιορισμένοι υπολογιστικοί πόροι που διαθέτουν, καθιστούν το έργο επεξεργασίας των δεδομένων αρκετά δύσκολο. Ακόμα, μια μονολιθική υποδομή επεξεργασίας δεδομένων αποτελεί μοναδικό σημείο αποτυχίας του συστήματος ενώ ταυτόχρονα υπάρχει πιθανότητα για κακόβουλη επίθεση, η οποία θέτει σε κίνδυνο την ιδιωτικότητα και ακεραιότητα των δεδομένων που έχουν συσσωρευτεί. Για παράδειγμα, μπορούμε να φανταστούμε έναν διακομιστή, ο οποίος συλλέγει τα δεδομένα που παράγουν διάφορες συσκευές έτσι ώστε να εκπαιδεύσει ένα νευρωνικό δίκτυο. Πέραν τούτου, η συλλογή ευαίσθητων πληροφοριών σε μια κεντρική δομή δεν είναι και η καλύτερη, ιδέα αφού δεν διατηρείται η ιδιωτικότητα των δεδομένων που παράχθηκαν στα άκρα του δικτύου. Το γεγονός αυτό μας ωθεί στην αναζήτηση μιας εναλλακτικής διαδικασίας εκπαίδευσης

μοντέλων χωρίς την συσσώρευση δεδομένων σε μια κεντρική οντότητα. Ένας μηχανισμός που μπορεί να βελτιώσει το πρόβλημα αυτό, είναι η τεχνική ομοσπονδιακής μάθησης (*Federated Learning*). Με την χρήση της τεχνικής αυτής η εκπαίδευση των μοντέλων γίνεται στα άκρα, γεγονός που σημαίνει ότι υπάρχει λιγότερος φόρτος επικοινωνίας με κάποιο κεντρικό διακομιστή, ενώ ταυτόχρονα τα δεδομένα διατηρούνται ασφαλή.

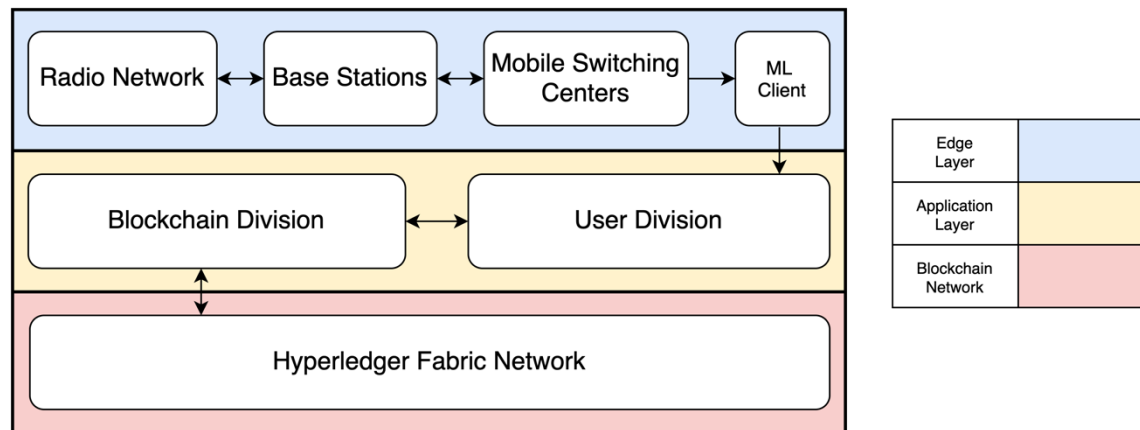
Ακόμη, ένα *Blockchain* μπορεί να συνεισφέρει στην αποκεντρωμένη αποθήκευση δεδομένων. Παρόλα αυτά, η τεχνολογία *Blockchain*, η οποία μετρά μόλις 12 χρόνια ζωής, μετά από την πρώτη υλοποίηση της στο δίκτυο οικονομικών συναλλαγών *Bitcoin*, παρουσιάζει προβλήματα όσον αφορά την κλιμακωσιμότητά της. Συγκεκριμένα, οι υφιστάμενες *Blockchain* αρχιτεκτονικές πάσχουν από κακές επιδόσεις όσον αφορά το *Throughput* και το *Latency*. Για παράδειγμα, το αντίστοιχο *Throughput* και *Latency* για το *Bitcoin* είναι 7 Tx/Sec και 10 λεπτά.

Επιπλέον, τίθενται ερωτήματα όσον αφορά τη διακυβέρνηση (*Governance*). Δηλαδή, μπορεί το σύστημα να είναι ευέλικτο σε μελλοντικές αλλαγές διατηρώντας την αξιοπιστία και την διαφάνεια του ως προς όλα τα μέλη του δικτύου; Ακόμη, σε πολλές περιπτώσεις θέλουμε να δίνουμε δικαιώματα στους χρήστες ενός *Blockchain*. Δηλαδή, ένας χρήστης, ίσως έχει την δυνατότητα να διαβάσει από ένα καθολικό (*Ledger*) αλλά να μην μπορεί να γράψει σε αυτό. Όλες οι προαναφερθείσες προκλήσεις, αποτελούν ένα υποσύνολο των προβλημάτων που λήφθηκαν υπόψη κατά την δημιουργία της *TriaBase*.

1.3 Συνεισφορά

Η συνεισφορά της εργασίας αυτής αφορά την πρόταση και αποτίμηση μιας ιδιωτικής αλυσίδας συστοιχιών με δικαιώματα, (*Permissioned Blockchain*) *IoT* βάσης δεδομένων, η οποία πλαισιώνεται από την τεχνική *Federated Learning*, για εκπαίδευση μοντέλων μηχανικής μάθησης, στα άκρα. Το όνομα της προτεινόμενης αρχιτεκτονικής είναι *TriaBase*, το οποίο προέρχεται από τις λέξεις «Τρία» που παραπέμπει στο Web 3.0 (τρίτη γενεά Web η οποία θα είναι αποκεντρωμένη) και την λέξη «Base» που παραπέμπει στις Βάσεις δεδομένων. Σκοπός της αρχιτεκτονικής αυτής, είναι η βελτιωμένη επίδοση ενός *Blockchain* με στόχο το αυξημένο *Throughput*, ως επίσης την δημιουργία βέλτιστων στρατηγικών υποβολής επερωτήσεων που αφορούν *IoT* δεδομένα.

Η αρχιτεκτονική *TriaBase* που φαίνεται στο σχήμα 1.1 έχει δημιουργηθεί στην πρώτη της έκδοση, δεδομένης τοπολογίας ενός δικτύου τηλεπικοινωνιών.



Σχήμα 1.1 Αρχιτεκτονική *TriaBase*

Αρχικά να αναφέρουμε ότι η *TriaBase* χωρίζεται σε τρία κύρια μέρη. Το επίπεδο άκρων (*Edge Layer*), το επίπεδο εφαρμογής (*Application Layer*) και το δίκτυο Αλυσίδας Συστοιχιών (*Blockchain Network*). Στο πιο πάνω σχήμα, η ροή των δεδομένων γίνεται από πάνω προς τα κάτω, δηλαδή από το *Edge Layer*, όπου παράγονται τα δεδομένα, προς το *Blockchain Network*, όπου αποθηκεύονται.

Στο *Edge Layer* εντάσσονται διάφορες άλλες στρώσεις, όπως *Radio Network Layer*, οι *Base Station*, τα *Mobile Switching Centers* και οι *ML Clients* όπου εκπαιδεύονται τα *Machine learning* μοντέλα. Επίσης, το *Application Layer* χωρίζεται σε δύο μέρη, το Τμήμα Χρήστη (*User Division*) και το Τμήμα Αλυσίδας Συστοιχιών (*Blockchain Division*). Τελευταίο κομμάτι της αρχιτεκτονικής *TriaBase*, αποτελεί το *Blockchain Network*, το οποίο στην περίπτωση μας υλοποιείται από το *Blockchain framework* ανοικτού κώδικα, *Hyperledger Fabric*.

Αρχίζοντας από το *Edge Layer*, έχουμε τις κεραίες τηλεπικοινωνιών οι οποίες παράγουν συνεχώς δεδομένα με μεγάλη ταχύτητα. Τα δεδομένα αυτά μπορεί να αφορούν συχνότητες εκπομπής των κεραιών, τον αριθμό των *Hand Over* που έγιναν, τον αριθμό κλήσεων σε μια περιοχή και ούτω καθεξής. Τα δεδομένα αυτά μεταβιβάζονται στους τοπικούς *Base Station*, οι οποίοι ελέγχουν ένα αριθμό από κεραίες. Στη συνέχεια, τα δεδομένα καταλήγουν στα *Mobile Switching Centers*, όπου οργανώνονται σε αρχεία.

Στο δεύτερο επίπεδο το οποίο ονομάζεται *Application Layer*, καταλήγουν τα δεδομένα από το *Edge Layer* για μια συγκεκριμένη γεωγραφική περιοχή. Ουσιαστικά στο *Application Layer* ανήκουν όλοι οι τοπικοί κόμβοι της αρχιτεκτονικής *TriaBase*. Ένας κόμβος στην *TriaBase* χωρίζεται σε δύο μέρη, το *Blockchain* και το *User Division*.

Στο *User Division* μπορούμε να επεξεργαστούμε και να οργανώσουμε τα δεδομένα με τον τρόπο που μας βολεύει. Αναλυτικότερα, σε αυτό το σημείο λαμβάνει μέρος η διαδικασία εκπαίδευσης των *Machine learning* μοντέλων με την τεχνική *Federated Learning*. Συγκεκριμένα, ένας κόμβος στην *TriaBase* διαδραματίζει τον ρόλο «Server» και αναλαμβάνει την διεξαγωγή πολλαπλών γύρων εκπαίδευσης ενός νευρωνικού δικτύου. Σε συνέχεια ο «Server»/*User Division*, αναλαμβάνει να κάνει δομήσει ένα καθολικό *Machine learning* μοντέλο, βάση των μοντέλων που παράχθηκαν στο *Edge Layer*. Με άλλα λόγια ο κάθε *ML Client* που βρίσκεται στο *Edge Layer*, δημιουργεί και εκπαιδεύει ένα τοπικό μοντέλο το οποίο αποστέλλει στο *User Division* / «Server» του τοπικού κόμβου *TriaBase*. Τέλος το συγκεντρωτικό μοντέλο προωθείται στο *Blockchain Division* για αποθήκευση .

Το δεύτερο κομμάτι του *Application Layer* ονομάζεται *Blockchain Division* και φαίνεται στο σχήμα 1.1. Εδώ γίνεται η προσομοίωση/εκτέλεση (Simulation/Execution) μιας δοσοληψίας (*transaction*) βάση κάποιων έξυπνων συμβολαίων (*smart contract*). Ένα *smart contract* περιγράφει τις διεργασίες που αφορούν ένα αντικείμενο (*object*) στο *Blockchain*. Στην περίπτωση μας, όπου τα *objects* είναι μοντέλα *Machine learning*, θέλουμε να υπάρχει δυνατότητα εγγραφής και ανάκτησής τους από το καθολικό (*Ledger*). Αφού λοιπόν εκτελεστεί ο αντίστοιχος κώδικας σε ένα *smart contract* ένα *transaction* λαμβάνει την ανάλογη έγκριση (*endorsement*) και είναι έτοιμο να ενσωματωθεί στο επόμενο μπλοκ (*block*) , το οποίο θα διαδοθεί σε όλο το *Blockchain Network*. Επίσης, το *Blockchain Division* του κάθε κόμβου στην *TriaBase* διατηρεί ένα τοπικό αντίγραφο του *Ledger* με μοντέλα από όλο το δίκτυο.

Πέραν από την αποθήκευση νέων μοντέλων στην *TriaBase*, υπάρχει η δυνατότητα υποβολής ερωτήσεων. Συγκεκριμένα, η λειτουργία ανάκτησης πληροφοριών, στηρίζεται στα μοντέλα *Machine learning* και τις προβλέψεις τις οποίες κάνουν. Όταν ένας χρήστης χρειάζεται να αναζητήσει πληροφορίες για μια συγκεκριμένη γεωγραφική περιοχή, πρέπει να ανακτήσει το σχετικό μοντέλο και στην συνέχεια να υποβάλει μια ερώτηση σε αυτό. Όπως έχουμε αναφέρει στην αρχή αυτής της υποενότητας, η

TriaBase είναι ένα permissioned περιβάλλον. Γι' αυτό τον λόγο κάθε χρήστης που έχει πρόσβαση στο *Blockchain* πρέπει να είναι εξουσιοδοτημένος.

Για την υποβολή επερωτήσης οι χρήστες πρέπει να συνδεθούν στο *User Division* έτσι ώστε να δημιουργήσουν ένα query σχετικά με το μοντέλο που θέλουν να ανακτήσουν. Αφού γίνει η ανάκτηση του μοντέλου από το τοπικό *Ledger* του *Blockchain Division*, ο χρήστης μπορεί πλέον να υποβάλει σε αυτό μια πληθώρα από επερωτήσεις.

Τέλος, σχετικά με το κομμάτι του *Blockchain Network* όπως αυτό φαίνεται στο σχήμα 1.1, να αναφέρουμε ότι αυτό υλοποιείται από το *Hyperledger Fabric*. Λεπτομερέστερη περιγραφή του εργαλείου αυτού γίνεται στο κεφάλαιο 3 της παρούσας διπλωματικής εργασίας.

Συνοψίζοντας αυτή την υποενότητα, πρέπει να αναφέρουμε ότι η προτεινόμενη αρχιτεκτονική παρουσιάζει υποσχόμενες επιδόσεις όσον αφορά την βέλτιστη υποβολή επερωτήσεων και την ασφαλή διαχείριση δεδομένων. Όσον αφορά την εισαγωγή και διανομή μοντέλων υπάρχει κάποια συμφόρηση, αλλά αυτό είναι αναμενόμενο αφού η *TriaBase* βρίσκεται ακόμα σε αρχικά στάδια της ανάπτυξής της. Πέραν πάσης αμφιβολίας, η προσέγγιση αυτή αποτελεί μια καινοτόμο ιδέα, αφού στα χρονικά ενέπνευσής της, δεν υπάρχει κάποια παρόμοια πρόταση που να συνδυάζει την μηχανική μάθηση και την φύλαξη μοντέλων σε ένα κατανεμημένο σύστημα *Blockchain*. Το σενάριο χρήσης της *TriaBase* αφορά τον τομέα των τηλεπικοινωνιών, αλλά δυνητικά οι τομείς στους οποίους μπορούμε να επεκτείνουμε την λογική αυτή είναι πολλαπλοί και δεν περιορίζονται.

1.4 Περίγραμμα Εργασίας

Αυτή η διπλωματική εργασία αποτελείται από επτά κεφαλαία. Αυτά είναι η «Εισαγωγή», η «Σχετική Δουλειά», το «Τεχνολογικό υπόβαθρο», η «Αρχιτεκτονική *TriaBase*», η «Πειραματική μεθοδολογία», η «Πειραματική αποτίμηση» και τα «Συμπεράσματα».

Στο τρέχον κεφάλαιο έχουμε δει εισαγωγικά, τον λόγο για τον οποίο αποφασίσαμε να ερευνήσουμε αυτό το πεδίο και μερικά από τα σχετικά προβλήματα. Επίσης, είδαμε εν συντομία την συνεισφορά που έχει επιτευχθεί μέσα από την πρόταση της αρχιτεκτονικής *TriaBase*.

Στο δεύτερο κεφάλαιο ο αναγνώστης μπορεί να δει διάφορες δουλειές που έχουν γίνει για το πεδίο με το οποίο ασχολείται η έρευνά μας. Αυτές περιλαμβάνουν διάφορες αρχιτεκτονικές *Blockchain* και δημοσιεύσεις που αφορούν την εκπαίδευση νευρωνικών δικτύων για αποθήκευση *IoT* δεδομένων. Ακόμη σε αυτό το κεφάλαιο γίνεται αναφορά σε τεχνολογίες αιχμής οι οποίες χρησιμοποιούνται στην βιομηχανία για διαχείριση *IoT* δεδομένων.

Στο τρίτο κεφάλαιο δίνονται πληροφορίες σχετικές με το τεχνολογικό υπόβαθρο με το οποίο πρέπει να είναι γνώριμος ο αναγνώστης της διπλωματικής εργασίας. Το τεχνολογικό υπόβαθρο περιλαμβάνει την περιγραφή όρων όπως τα δίκτυα ομότιμων κόμβων (*Peer-to-Peer*), η κωδικοποίηση *Base64* και η μέθοδος κρυπτογράφησης με την χρήση δημόσιου και ιδιωτικού κλειδιού (*Public Private Key*). Επιπλέον, υπάρχει αναλυτική περιγραφή του *Blockchain framework Hyperledger Fabric*, με αναφορά στο μοντέλο εκτέλεσης *transactions – execute/order/validate* - και των κύριων εννοιών που το πλαισιώνουν, όπως τον ομότιμο κόμβο (*Peer*), την υπηρεσία σειριοποίησης (*Ordering Service*), έξυπνά συμβόλαια (*smart contracts or chaincode*) και το καθολικό (*Ledger*).

Στο τέταρτο κεφάλαιο παρουσιάζεται η αρχιτεκτονική *TriaBase*. Συγκεκριμένα στο κεφάλαιο αυτό περιγράφονται οι στρώσεις της αρχιτεκτονικής και οι διεργασίες που γίνονται σε κάθε επίπεδο. Όσον αφορά το *User Division* περιγράφονται οι αλγόριθμοι εισαγωγής και υποβολής επερωτήσεων και γίνεται αναφορά στον τρόπο λειτουργίας της τεχνικής *Federated Learning*. Τέλος, όσον αφορά το *Blockchain Division* περιγράφονται τα αντίστοιχα *smart contracts* που έχουν υλοποιηθεί.

Η πειραματική μεθοδολογία σκιαγραφείται στο πέμπτο κεφάλαιο, όπου ο αναγνώστης μπορεί να δει τα πειράματα που προτείνουμε για την αξιολόγηση της *TriaBase*, όσον αφορά το *Throughput* και το *Latency*. Ακόμη, σε αυτό το κεφάλαιο περιγράφεται η φύση των δεδομένων που χρησιμοποιήθηκαν, ως επίσης το δίκτυο που στήθηκε για τα πειράματά μας.

Στο έκτο κεφάλαιο, αναλύονται τα αποτελέσματα των πειραμάτων που περιγράψαμε. Αυτά αφορούν την εισαγωγή δεδομένων και υποβολή επερωτήσεων. Ακόμη σε αυτό το κεφάλαιο αντιπαραβάλλουμε την προσέγγιση της *TriaBase* με την παραδοσιακή προσέγγιση εγγραφής δεδομένων σε ένα *Blockchain*, όπου κάθε εγγραφή αποτελεί ένα διαφορετικό *transaction*.

Τέλος, το κεφάλαιο επτά αναφέρεται στα γενικά συμπεράσματα της διπλωματικής εργασίας, καθώς και την μελλοντική δουλειά που μπορεί να γίνει σε ερευνητικό επίπεδο για την περεταίρω ανάπτυξη της *TriaBase*.

Κεφάλαιο 2

Σχετική Δουλειά

2.1 Είδη Αλυσίδων Συστοιχιών	10
2.1.1 Δημόσιες	10
2.1.2 Ιδιωτικές.....	11
2.1.3 Με Παροχή Άδειας	11
2.1.4 Χωρίς Χρήση Άδειας.....	12
2.2 Ερευνητικές Τεχνολογίες Αλυσίδων Συστοιχιών.....	12
2.2.1 AnyLog: A Grand Unification of the Internet of Things.....	12
2.1.2 CAPER: A Cross-Application Permissioned <i>Blockchain</i>	15
2.3 <i>Influxdb</i>: Μια Βάση Δεδομένων για Χρονοσειρές	20
2.3.1 Βασικές Έννοιες	20
2.3.2 Εσωτερικοί Μηχανισμοί της Βάσης	21
2.4 Διαχείριση Δεδομένων Διαδικτύου Αντικειμένων με το Ενδιάμεσο Λογισμικό Node-RED.....	22
2.5 Αποσύνθεση Μεγάλων Δεδομένων	23
2.6 Τεχνική Ομοσπονδιακής Μάθησης.....	26

Σε αυτό το κεφάλαιο θα δούμε εν συντομία διάφορες τεχνολογίες και ακαδημαϊκά άρθρα, τα οποία μελετήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας. Η σχετική μελέτη που έγινε αρχίζει από πιο γενικά θέματα όπως την μελέτη διάφορων ειδών αλυσίδων συστοιχιών (*Blockchain*) και καταλήγει σε πιο ειδικά θέματα όπως την αποδοτική διαχείριση δεδομένων με την χρήση μοντέλων μηχανικής μάθησης (*Machine*

learning). Κομμάτι της σχετικής δουλειάς που θα περιγράψουμε πιο κάτω, έχει χρησιμοποιηθεί ως ερέθισμα για την έμπνευση της αρχιτεκτονικής *TriaBase*, όπως αυτή θα περιγραφεί στο κεφάλαιο 4.

2.1 Είδη Αλυσίδων Συστοιχιών

Στις μέρες μας, δώδεκα χρόνια μετά την πρώτη *Blockchain* εφαρμογή, το *Bitcoin* [1] [2] [3], οι τεχνολογικές εξελίξεις και οι ανάγκες της βιομηχανίας, έχουν οδηγήσει την επιστημονική κοινότητα στην δημιουργία διάφορων ειδών *Blockchains*. Πιο αναλυτικά έχουμε τέσσερεις κατηγορίες *Blockchain* οι οποίες εξυπηρετούν και διαφορετικό σκοπό. Στον πίνακα 2.1 βλέπουμε παραδείγματα *Blockchain* εργαλείων, ανάλογα με την κατηγορία στην οποία ανήκουν.

	Permissionless	Permissioned
Public	<i>Bitcoin</i>	<i>Hyperledger Sawtooth</i>
Private	LTO Network	<i>Hyperledger Fabric</i>

Πίνακας 2.1 Κατηγορίες Blockchain

2.1.1 Δημόσιες

Οι Δημόσιες Αλυσίδες Συστοιχιών (*Public Blockchains*) [4] [5] [6], αποτελούν την πιο διαδεδομένη κατηγορία *Blockchain*, με το *Bitcoin* να είναι το πρώτο και πιο δημοφιλές. Ειδικότερα, με τον όρο Public εννοούμε ότι το δίκτυο είναι ανοικτό προς όλους και οποιοσδήποτε μπορεί να συμμετάσχει σε αυτό. Ένα από τα κύρια χαρακτηριστικά αυτής της δομής, είναι ότι παρέχει πλήρη ανωνυμία, ενώ ταυτόχρονα το σύστημα είναι σε μεγάλο βαθμό αποκεντρωμένο, χωρίς συγκεκριμένη οντότητα να ελέγχει το δίκτυο. Εντούτοις, ένα *Public Blockchain* αντισταθμίζει τα προαναφερθέντα πλεονεκτήματα που

προσφέρει με πολλά μειονεκτήματα. Τα μειονεκτήματα αυτά σχετίζονται κυρίως με την δυσκολία κλιμακωσιμότητας και κακής επίδοσης. Αυτό οφείλεται στο γεγονός ότι για να διασφαλίσουμε την ασφαλή λειτουργία του συστήματος [5] χρησιμοποιούνται πιθανοτικοί αλγόριθμοι ομοφωνίας (*consensus*) [7]. Οι αλγόριθμοι αυτοί χρειάζονται εκτεταμένους υπολογιστικούς πόρους, με αποτέλεσμα το δίκτυο να είναι αργό, και να γίνεται μεγάλη σπατάλη ενέργειας [8].

2.1.2 Ιδιωτικές

Μια Ιδιωτική Αλυσίδα Συστοιχιών (*Private Blockchain*) [4] [5] [6] αποτελεί ένα κλειστό περιβάλλον μεταξύ διάφορων συμμετεχόντων. Ειδικότερα, οι οντότητες που συμμετέχουν σε αυτό, είναι συγκεκριμένες και για να εισέλθει μια καινούρια οντότητα στο δίκτυο, πρέπει να δοθεί άδεια από τους διαχειριστές του δικτύου. Επιπρόσθετα, το καθολικό (*Ledger*) που διατηρεί ένα τέτοιο *Blockchain* είναι ιδιωτικό και δεν μπορεί να έχει πρόσβαση σε αυτό οποιοσδήποτε. Επιπλέον, ένα *Private Blockchain* μπορεί να λειτουργήσει με ντετερμινιστικούς *consensus* αλγόριθμους, οι οποίοι είναι σαφώς πιο φτηνοί όσον αφορά τον υπολογιστικό φόρτο. Τα προαναφερθέντα χαρακτηριστικά επιτρέπουν μεγάλη κλιμακωσιμότητα και καλύτερες επιδόσεις σε σχέση με τα *Public Blockchains*.

2.1.3 Με Παροχή Άδειας

Αλυσίδα Συστοιχιών με δικαιώματα (*Permissioned Blockchain*) [4] [5] [6] είναι ειδικά διαμορφωμένα συστήματα, τα οποία ελέγχουν τα δικαιώματα του κάθε χρήστη. Συγκεκριμένα, μπορεί σε έναν οργανισμό, συγκεκριμένοι χρήστες να έχουν την δυνατότητα εγγραφής στο *Ledger*, ενώ κάποιιοι άλλοι να μπορούν μόνο να διαβάζουν από αυτό. Στις πλείστες περιπτώσεις τα *Private Blockchains* είναι και *Permissioned*, αφού οι δύο έννοιες είναι άμεσα συνδεδεμένες.

2.1.4 Χωρίς Χρήση Άδειας

Με τον όρο «Χωρίς Χρήση Άδειας» (*Permissionless*), εννοούμε ότι δεν υπάρχουν περιορισμοί πρόσβασης και αλληλεπίδρασης σε ένα σύστημα *Blockchain*. Ένα τέτοιο παράδειγμα είναι το *Bitcoin* και το *Ethereum* [9]. Στις πλείστες περιπτώσεις τα *Public Blockchains* είναι κα *Permissionless*, αφού οι δύο έννοιες είναι αλληλένδετες.

2.2 Ερευνητικές Τεχνολογίες Αλυσίδων Συστοιχιών

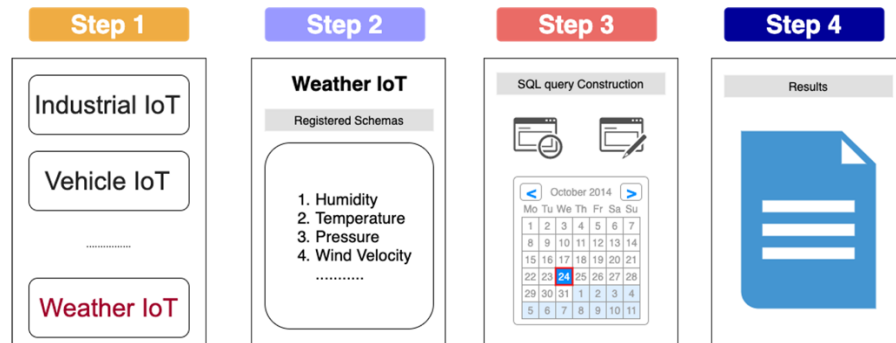
Στα πλαίσια της προσπάθειας μας να προσανατολιστούμε στον χώρο του *Blockchain*, μελετήθηκαν διάφορες δουλειές που έγιναν σε ακαδημαϊκό επίπεδο. Συγκεκριμένα είδαμε *Blockchain* για διαχείριση *IoT* [10] δεδομένων ως επίσης *Blockchain* στα οποία μπορούν να τρέχουν πολλαπλές εφαρμογές σχηματίζοντας παράλληλα *Ledgers*.

2.2.1 AnyLog: A Grand Unification of the Internet of Things

Στις μέρες μας η χρήση των *IoT* συσκευών γίνεται όλο και πιο διαδεδομένη. Οι συσκευές αυτές παράγουν ένα τεράστιο όγκο δεδομένων, ο οποίος πρέπει να αποθηκεύεται, να διανέμεται και να τυγχάνει επεξεργασίας. Η χρήση *SQL* βάσεων δεδομένων σε συγκεκριμένες περιπτώσεις δεν είναι κατάλληλη. Έτσι λοιπόν ένα μέρος της επιστημονικής κοινότητας έχει στρέψει την προσοχή του σε νέες τεχνολογίες όπως το *Blockchain*. Ένα εργαλείο *Blockchain* το οποίο έχει προταθεί για την διαχείριση *IoT* δεδομένων είναι το AnyLog [11].

Το AnyLog είναι ένα *Public/Permissionless Blockchain* το οποίο δημιουργήθηκε ειδικά για την διαχείριση *IoT* δεδομένων. Δυνητικά, το AnyLog επιτρέπει σε διάφορες συσκευές να ενταχθούν σε ένα δίκτυο όπου δημοσιοποιούν τα δεδομένα τα οποία παράγουν, με απώτερο σκοπό να μπορούν να έχουν πρόσβασή σε αυτά διάφοροι χρήστες. Φυσικά, οι συσκευές που παράγουν δεδομένα, λαμβάνουν κάποια ανταμοιβή για τις υπηρεσίες τους. Με αυτό τον τρόπο οι χρήστες του δικτύου παρακινούνται να συμμετάσχουν και να συνεισφέρουν σε ένα τέτοιο δίκτυο. Επιπλέον, μια από τις μεγαλύτερες προκλήσεις που επιλύει το AnyLog είναι ότι προσφέρει μια ειδικά διαμορφωμένη διεπιφάνεια η οποία επιτρέπει στους χρήστες να υποβάλλουν στο *Blockchain* ειδικά διαμορφωμένες

επερωτήσεις. Στο σχήμα 2.1 φαίνεται αναλυτικότερα ο τρόπος με τον οποίο ένας χρήστης μπορεί να αλληλεπιδράσει με το AnyLog.



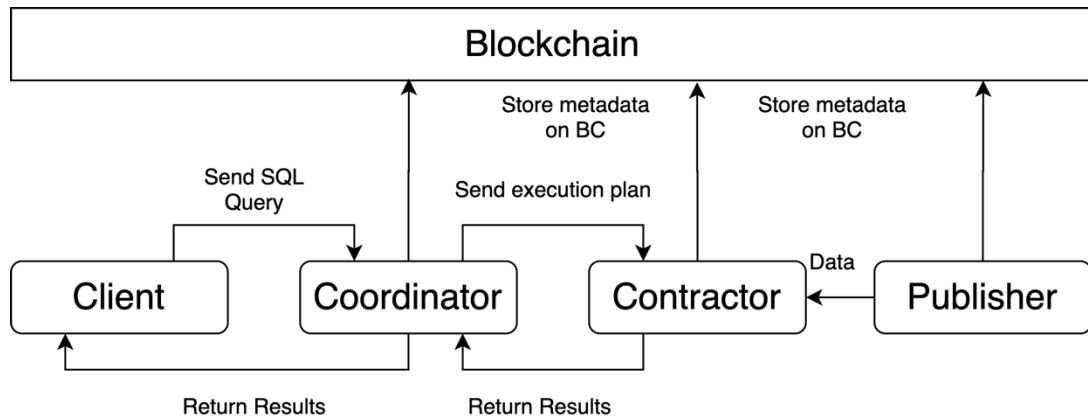
Σχήμα 2.1 Διαδικασία κατασκευής query

Αρχικά ο χρήστης μπορεί να επιλέξει τον τομέα, ο οποίος τον ενδιαφέρει. Για παράδειγμα η αναζήτηση του μπορεί να εστιάζει σε βιομηχανικά δεδομένα *IoT*. Στο επόμενο βήμα ο χρήστης θα μπορεί να επιλέξει τη δομή που τον εκφράζει. Στο τρίτο στάδιο της διαδικασίας, το σύστημα βοηθά τον χρήστη να σχηματίσει μια επερώτηση (*SQL query*) και να το υποβάλει στο *Blockchain*. Τέλος, τα αποτελέσματα επιστέφονται στον χρήστη για κατανάλωση.

Η αρχιτεκτονική του AnyLog, βασίζεται σε διάφορα είδη κόμβων και δομές δεδομένων. Πιο κάτω βλέπουμε τις πιο σημαντικές οντότητες του δικτύου και τις ιδιότητές τους.

- *Client*: Παράγουν *SQL queries* και καταναλώνουν τα αποτελέσματά τους.
- *Coordinator*: Είναι κόμβοι που παραλαμβάνουν τα queries, τα βελτιστοποιούν και ετοιμάζουν ένα πλάνο εκτέλεσης (*execution plan*), το οποίο προωθούν στους *Contractors* για εκτέλεση
- *Contractor*: Είναι κόμβοι που αποθηκεύουν τα δεδομένα που τους στέλνουν οι *Publishers* και εκτελούν *query execution plans* που τους αποστέλλουν οι *Coordinators*
- *Publisher*: Οι *Publisher* παράγουν *IoT* δεδομένα και τα προωθούν στους *Contractors* για αποθήκευση
- *Blockchain*: Είναι μια δομή δεδομένων που διατηρεί μεταδομένα των δοσοληψιών, ως επίσης τις ρυθμίσεις του δικτύου

Αναλυτικότερα, η ροή δεδομένων μεταξύ των πιο πάνω οντοτήτων παρουσιάζεται διαγραμματικά στο σχήμα 2.2



Σχήμα 2.2 AnyLog Execution Flow

1. Αρχικά οι *Publishers* αποστέλλουν τα δεδομένα που παράγουν οι *IoT* συσκευές, στα άκρα του δικτύου, στους *Contractors* οι οποίοι αποθηκεύουν τα δεδομένα αυτά.
2. Ο *Client* υποβάλλει ένα query σε κάποιο *Coordinator*.
3. Ο *Coordinator*, παράγει το πλάνο εκτέλεσης για το query, το οποίο έχει παραλάβει.
4. Το πλάνο εκτελείται σε όλους τους σχετικούς *Contractors* για να υπολογιστεί η απάντηση. Τα αποτελέσματα που παράγονται επιστρέφονται στους *Coordinators*.
5. Οι *Coordinators* ομαδοποιούν τα δεδομένα που έχουν παραλάβει και αποστέλλουν το σχετικό αποτέλεσμα στους *Clients*.
6. Τέλος, οι *Clients* πληρώνουν κάποιο ποσό στους *Coordinators* για την υπηρεσία που τους πρόσφεραν. Οι *Coordinators* διαδοχικά αναλαμβάνουν να ανταμείψουν τους *Contractors* και οι *Contractors* με την σειρά τους τους *Publishers*.

Όπως έχει προαναφερθεί, το AnyLog, παρέχει έναν μηχανισμό ανταμοιβής των μελών που συμμετέχουν στο δίκτυο. Πιο κάτω αναγράφονται οι σχετικές πληροφορίες ανταμοιβής για την συνεισφορά του κάθε μέλους.

- *Data Publication*: Όταν ένας *Client* ή *Contractor* έχει πρόσβαση στα δεδομένα τα οποία έχει παράξει ο *Publisher*.

- *Data Storage*: Ο *Contractor* ανταμείβεται από τον *Publisher* για να αποθηκεύει τα δεδομένα του.
- *Data Processing / Contractor*: Ο *Contractor* ανταμείβεται βάση της υπολογιστικής δύναμης που έχει χρησιμοποιήσει για να επεξεργαστεί κάποιο query.
- *Data Processing / Coordinator*: Ο *Coordinator* ανταμείβεται για το query optimization και την επεξεργασία την οποία κάνει.

2.1.2 CAPER: A Cross-Application Permissioned Blockchain

Το CAPER [12], είναι ένα εργαλείο το οποίο εντάσσεται στην κατηγορία των Permissioned Blockchain. Οι ερευνητές που ανέπτυξαν το CAPER πραγματεύονται το σενάριο χρήσης πολλαπλών Applications σε ένα περιβάλλον Blockchain ενώ ταυτόχρονα εξετάζουν διάφορους consensus αλγόριθμους και πως αυτοί επηρεάζουν την επίδοση του συστήματος.

Αρχικά, θα αναφερθούμε στην έννοια του Application. Ένα Application αποτελεί την επιχειρηματική λογική που ακολουθούν όσοι συμμετέχουν σε ένα δίκτυο Blockchain.

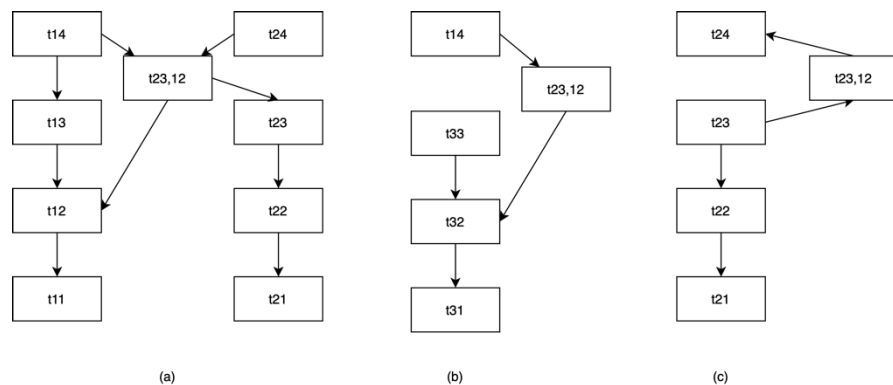
Για να καταλάβουμε όμως καλύτερα τον ορισμό αυτό, ας φέρουμε για παράδειγμα ένα εργοστάσιο το οποίο συμμετέχει σε μια αλυσίδα ανεφοδιασμού (Supply Chain), Blockchain περιβάλλον. Το εργοστάσιο μπορεί να τρέχει ταυτόχρονα διάφορες εφαρμογές (πχ εφαρμογή αποθέματος, εφαρμογή προμηθευτών, εφαρμογή πελατών κλπ), οι οποίες χρειάζεται να αλληλοεπιδρούν με το Blockchain συνεχώς, έτσι ώστε να καταχωρούν τις δοσοληψίες τις οποίες κάνουν. Οι δοσοληψίες αυτές μπορεί να λαμβάνουν χώρο σε τοπικό επίπεδο (local level), δηλαδή αφορούν τοπικά δεδομένα της ίδιας της εφαρμογής, ή να είναι σε καθολικό επίπεδο (global level) και να συσχετίζονται με δεδομένα άλλων εφαρμογών του δικτύου.

Σε αυτό το σημείο δημιουργούνται βέβαια και πολλές προκλήσεις. Συγκεκριμένα, όταν έχουμε πολλαπλές εφαρμογές, υπάρχει ανάγκη να διατηρούμε τα δεδομένα που παράγουν ξεχωριστά. Δηλαδή, αν για παράδειγμα υπήρχε ένα καθολικό Blockchain για όλες τις εφαρμογές, αυτό θα ήταν καταστροφικό, αφού σε ένα Permissioned περιβάλλον δεν πρέπει όλα τα δεδομένα και δοσοληψίες να είναι ορατές σε όλους. Επιπλέον, έχουμε

δοσοληψίες οι οποίες αφορούν δύο ή περισσότερες εφαρμογές. Αυτές οι δοσοληψίες αποτελούν ειδικές περιπτώσεις και πρέπει να είναι ορατές από διαφορετικές εφαρμογές.

Οι δημιουργοί του *CAPER* έλαβαν υπόψη τους τα ζητήματα αυτά έτσι ώστε να ξεπεράσουν τα προαναφερθέντα προβλήματα. Συγκεκριμένα, επιμελήθηκαν τα ακόλουθα.

- *Blockchain*: Χρήση ξεχωριστού *Blockchain* για κάθε εφαρμογή. Έτσι διατηρείται ο *Permissioned* Χαρακτήρας του συστήματος.
- *Transaction Batching*: Η διαδικασία ομαδοποίησης των συναλλαγών σε *block* μερικές φορές μπορεί να είναι χρονοβόρα και να απαιτεί την χρήση εκτεταμένων υπολογιστικών πόρων. Για τον λόγο αυτό το μοντέλο του *CAPER* προβλέπει ότι σε κάθε *block*, θα υπάρχει μόνο μια δοσοληψία.
- *Ledger Properties*: Όλες οι δοσοληψίες που καταγράφονται στο *Blockchain* πρέπει να είναι σε σειρά, ανάλογα με την χρονοσφραγίδα τους. Επιπλέον, δοσοληψίες που αφορούν πολλές εφαρμογές, δηλαδή είναι *Cross-Application Transactions* [12], συνδέονται με τις αντίστοιχες δοσοληψίες σε άλλα *Blockchains* χρησιμοποιώντας κρυπτογραφικούς συνδέσμους *Hash*.
- *DAG*: Για την υλοποίηση παράλληλων *Blockchains* που συνδέονται μεταξύ τους σε συγκεκριμένα σημεία, χρησιμοποιήθηκε μια ιδιαίτερη δομή δεδομένων, που ονομάζεται κατευθυνόμενος μη κυκλικός γράφος (*Directed Acyclic Graph -DAG*) [13]. Στο σχήμα 2.3 βλέπουμε την αναπαράσταση μιας τέτοιας δομής.

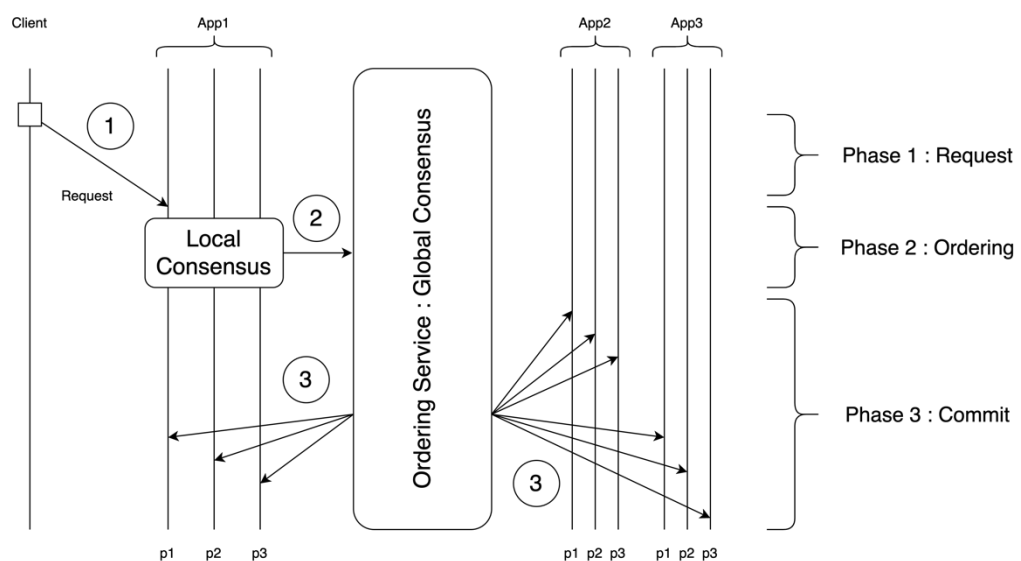


Σχήμα 2.3 Παράλληλα *Blockchain* με την χρήση *DAG*

Το (α) στο πιο πάνω σχήμα μας δείχνει πως συνδέονται τα *transaction* τριών διαφορετικών *Blockchain*. Για παράδειγμα η δοσοληψία $t_{23,12}$ σχετίζεται με τις δοσοληψίες t_{12} και t_{23} και συνεπώς συμπεριλαμβάνεται σε όλα τα *Blockchain* όπως φαίνεται στα μέρη (b) , (c) αντίστοιχα.

Επιπρόσθετα, στην δουλειά τους, οι συγγραφείς του *CAPER* προτείνουν τρεις διαφορετικούς αλγόριθμους ομοφωνίας (*Consensus*), οι οποίοι πιθανόν να αυξήσουν την απόδοση του συστήματος. Οι αλγόριθμοι αυτοί είναι οι ακόλουθοι :

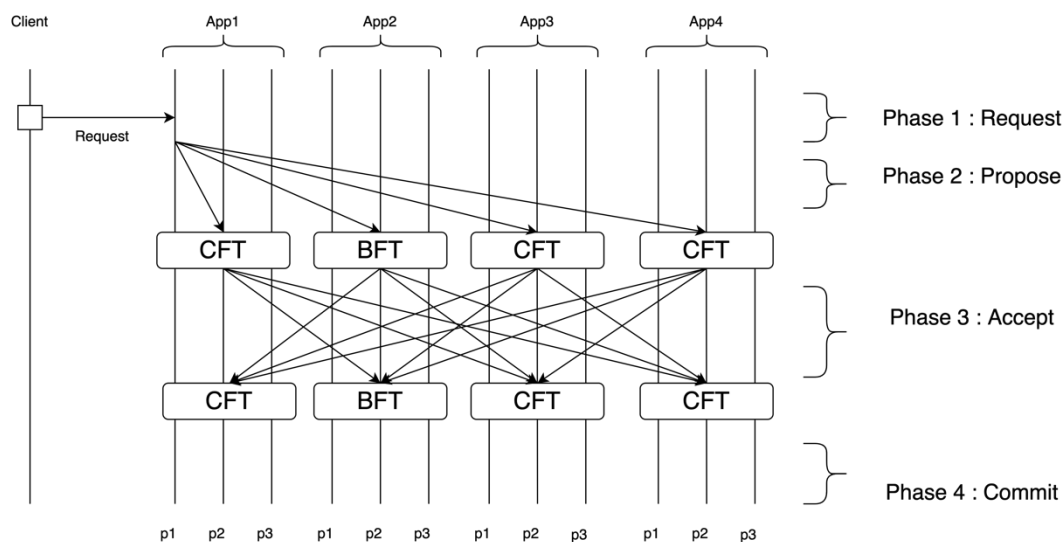
- *Global Consensus Using Orderers*: Όπως φαίνεται και στο σχήμα 2.4 αυτός ο αλγόριθμος χωρίζεται σε τέσσερις κυρίως φάσεις.
 1. Αρχικά ο *Client* αποστέλλει το *request* που αφορά μια συγκεκριμένη εφαρμογή στο *Primary Peer*. Τότε ενεργοποιείται τοπικός *consensus* αλγόριθμος, για να συμφωνήσουν σε μια κοινή τιμή, όλα τα *Peers* στα οποία τρέχει η συγκεκριμένη εφαρμογή.
 2. Στη συνέχεια, αφού ληφθεί τοπικά μια κοινή απόφαση ένα νέο *request* προωθείται στους *Orderers*, οι οποίοι αναλαμβάνουν την δουλειά του *Global Consensus* λαμβάνοντας υπόψη δοσοληψίες που προκύπτουν από άλλες εφαρμογές.
 3. Στο τρίτο στάδιο του αλγόριθμου τα *block* με τις δοσοληψίες διανέμονται σε όλα τα *Peers* του δικτύου για να εκτελεστούν τοπικά.
 4. Το *Block* προστίθεται στο *Ledger* κάθε *Peer*.



Σχήμα 2.4 *Global Consensus using Orderers*

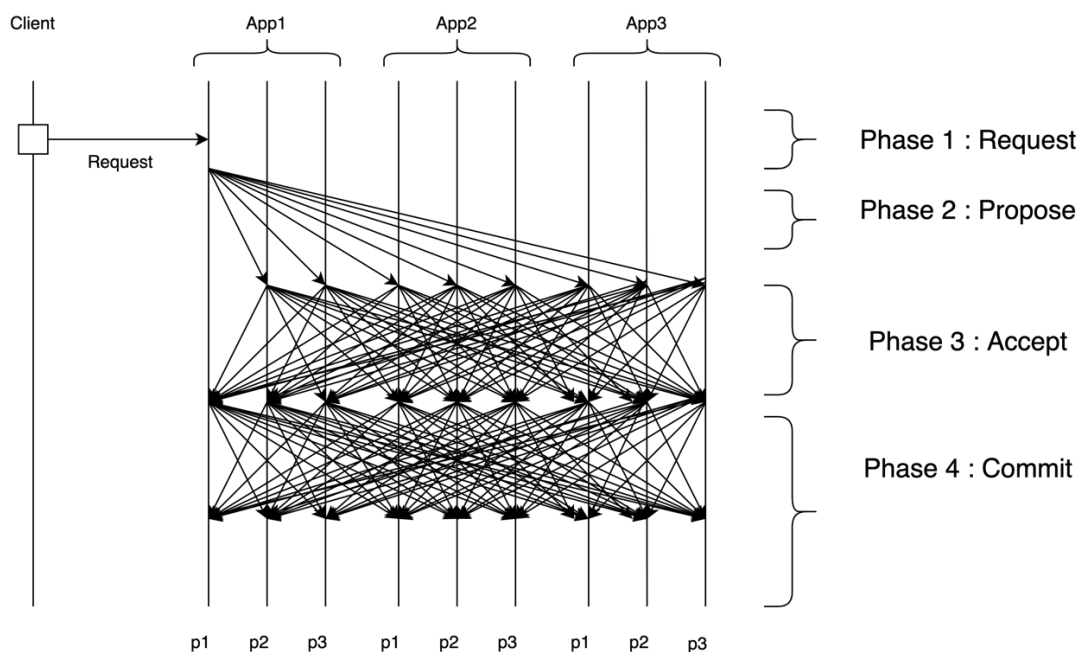
- Hierarchical Global Consensus: Όπως φαίνεται και στο σχήμα 2.5 αυτός ο αλγόριθμος χωρίζεται σε τρία στάδια.

1. Αρχικά ο *Client* αποστέλλει στο *Primary Peer* ένα *request* που αφορά μια συγκεκριμένη εφαρμογή. Το *Primary Peer* αναλαμβάνει να δημιουργήσει και να προωθήσει ένα *Proposal* σε κάθε *Primary Peer* όλων των εφαρμογών του δικτύου, το οποίο οργανώνει το τοπικό *consensus* κάθε εφαρμογής. Το *Proposal* αυτό εκτελείται τοπικά από κάθε εφαρμογή ξεχωριστά και αποστέλλεται απάντηση στις υπόλοιπες εφαρμογές του δικτύου. Αφού γίνει αυτή η διαδικασία και υπάρξει πλειοψηφική απόφαση μεταξύ όλων των εφαρμογών, προχωρούμε σε γύρο ανταλλαγής μηνυμάτων αποδοχής.
2. Όταν κάποιο *Peer* λάβει μήνυμα αποδοχής από την πλειοψηφία των εφαρμογών, είναι έτοιμο να προχωρήσει στην τοπική εκτέλεση των δοσοληψιών και στο τελικό τους *commit*.
3. Το *block*, με τις δοσοληψίες που έγιναν *commit*, προστίθεται στο *Ledger* του *Peer*.



Σχήμα 2.5 Hierarchical Global Consensus

- *One Level Global Consensus*: Όπως φαίνεται και στο σχήμα 2.6 αυτός ο αλγόριθμος χωρίζεται σε τρία στάδια.
 1. Αρχικά ο *Client* αποστέλλει κάποιο *request* σε όλα τα *Peers* του δικτύου, ανεξάρτητα από την εφαρμογή την οποία τρέχει σε αυτά. Αφού το *Peer* παραλάβει την προτεινόμενη δοσοληψία αναμεταδίδει στο δίκτυο ένα νέο μήνυμα αποδοχής
 2. Όταν κάποιο *Peer* λάβει μήνυμα αποδοχής από την πλειοψηφία των *Peers* στο δίκτυο, προχωρά στην τοπική εκτέλεση των δοσοληψιών και στο τελικό τους *commit*.
 3. Το *block*, με τις δοσοληψίες που έγιναν *commit*, προστίθεται στο *Ledger* του *Peer*.



Σχήμα 2.6 *One level Global Consensus*

Τα πειράματα που έγιναν σε αυτή την δουλειά, δείχνουν ότι σε γενικές γραμμές η χρήση του αλγόριθμου *Global Consensus Using Orderers*, είναι η πιο αποδοτική αφού επιτρέπει στο σύστημα να διατηρεί σχετικά μεγάλο *Throughput* και μικρό *Latency* [4] για μεγάλο αριθμό εισερχόμενων δοσοληψιών σε παράλληλα *Blockchain* που φιλοξενούν διαφορετικές εφαρμογές.

2.3 Influxdb: Μια Βάση Δεδομένων για Χρονοσειρές

Στην έρευνα που έγινε κατά την διάρκεια της παρούσας διπλωματικής εργασίας, εντάσσεται και η μελέτη βάσεων δεδομένων. Συγκεκριμένα, μελετήθηκε η περίπτωση μιας σχετικά καινούριας βάσης δεδομένων της *Influxdb* [14]. Η *Influxdb* είναι ένα έργο ανοικτού κώδικα και αφορά δεδομένα χρονοσειρών. Τέτοια δεδομένα μπορεί να είναι μετρήσεις από έναν μετεωρολογικό σταθμό, οι τιμές των μετοχών στο χρηματιστήριο και γενικότερα ότι δεδομένα παράγουν οι *IoT* συσκευές. Σε αυτή την υποενότητα, θα δούμε τις βασικές έννοιες της *Influxdb* έτσι ώστε να καταλάβουμε καλύτερα τον τρόπο με τον οποίο λειτουργεί.

2.3.1 Βασικές Έννοιες

- *Measurement*: *Measurement* ή αλλιώς μέτρηση είναι το όνομα που δίνουμε για να περιγράψουμε τα δεδομένα μας. Μπορούμε να φανταστούμε το *Measurement* ως αντίστοιχο του Πίνακα σε μια *SQL* βάση δεδομένων. Για παράδειγμα στο σχήμα 2.8 το *Measurement* αναγράφεται στην πρώτη γραμμή και έχει την τιμή «census».
- *Point*: Ένα *Point* αντιπροσωπεύει μια μεμονωμένη εγγραφή δεδομένων η οποία αποτελείται από τρία κυρίως μέρη. Τα μέρη αυτά είναι τα *Tags*, τα *Fields* και μια συγκεκριμένη χρονοσφραγίδα. Κάθε *Point* αναγνωρίζεται ως μοναδικό βάσει της χρονοσφραγίδας του. Παράδειγμα ενός ολοκληρωμένου *Point* βλέπουμε στο σχήμα....
- *Tags & Fields*: Τα *Tags* και τα *Fields* αντιπροσωπεύουν της στήλες ενός Πίνακα με *Records* σε μια αντίστοιχη *SQL* βάση δεδομένων. Η διαφορά την οποία έχουν τα *Tags* και τα *Fields* είναι ότι τα *Tags* χρησιμοποιούνται για την δημιουργία ευρετηρίων που αφορούν το συγκεκριμένο *Measurement*, ενώ τα *Fields* όχι. Για παράδειγμα, στο σχήμα 2.8 δύο πιθανές στήλες που θα μπορούσαν να θεωρηθούν *Tags* είναι το «location» και «scientist».

name: census				

time	butterflies	honeybees	location	scientist
2015-08-18T00:00:00Z	1	30	1	perpetua

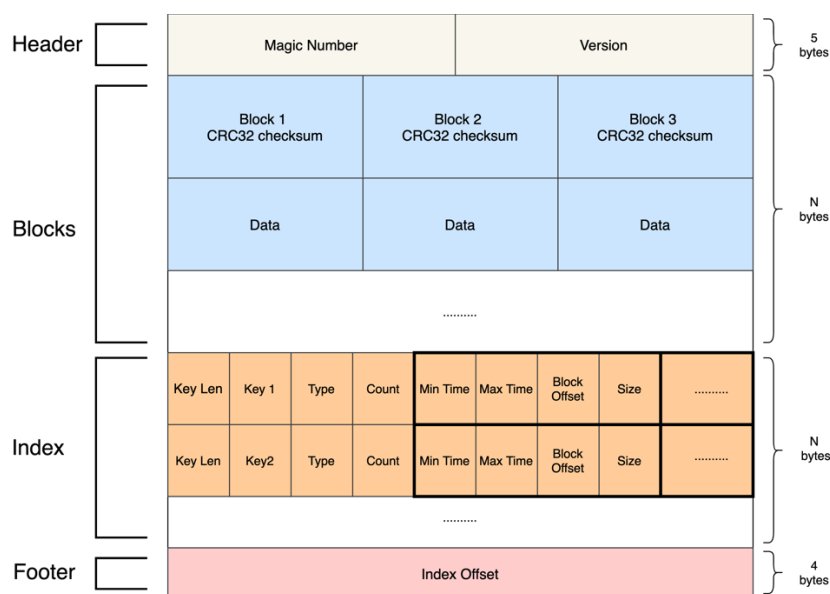
Σχήμα 2.7 Παράδειγμα Point στην Influxdb

2.3.2 Εσωτερικοί Μηχανισμοί της Βάσης

Όπως κάθε βάση δεδομένων, έτσι και η *Influxdb*, αποτελείται από διάφορα κομμάτια τα οποία σχηματίζουν το storage engine. Ειδικότερα, το κυρίως μέρος της *Influxdb* αποτελείται από τέσσερα δομικά στοιχεία τα οποία είναι το *Write Ahead Log*, η πτητική μνήμη *Cache*, τα *TSM* αρχεία και τα ευρετήρια *TSI*.

- *Write Ahead Log*: Ο κανόνας «γράφω πρώτα» write ahead, γνωστός και ως write ahead log [15] [14] είναι η μια logging στρατηγική όπου το log της βάσης δεδομένων ενημερώνεται πριν γίνει το τελικό commit στο δίσκο. Συγκεκριμένα, μια νέα εγγραφή προστίθεται πρώτα στο WAL αρχείο και στην συνέχεια χρησιμοποιείται το system call *fsync()* για να εγγραφούν τα δεδομένα στην δευτερεύουσα μνήμη. Ταυτόχρονα, ενημερώνεται η μνήμη cache. Αφού τα δεδομένα εγγραφούν στον δίσκο με επιτυχία, ένα μήνυμα επιβεβαίωσης επιστρέφεται. Το πρωτόκολλο WAL είναι χρήσιμο σε περίπτωση αποτυχίας της βάσης δεδομένων. Δηλαδή αν για οποιοδήποτε λόγο η βάση δεν καταφέρει να αποθηκεύσει τα δεδομένα στην δευτερεύουσα μνήμη, μπορεί να δοκιμάσει να επαναλάβει την εγγραφή ανατρέχοντας στο WAL αρχείο.
- *Cache*: Η μνήμη cache είναι ένας μικρός volatile αποθηκευτικός χώρος [14], ο οποίος χρησιμοποιείται για την γρήγορη και αποδοτική εκτέλεση queries. Στην μνήμη cache μπορεί να βρίσκονται δεδομένα στα οποία είχαμε πρόσβαση πρόσφατα ή δεδομένα που μόλις έχουν εγγραφεί στην βάση. Όταν ένα νέο query υποβληθεί στο σύστημα, τα δεδομένα στην μνήμη cache συνδυάζονται με δεδομένα από τον δίσκο, έτσι ώστε να δοθεί μια απάντηση. Σημαντικό είναι να αναφέρουμε ότι όταν εκτελείται ένα query τα δεδομένα από τον δίσκο αντιγράφονται στην cache έτσι ώστε τα αποτελέσματα να μην επηρεάζονται από νέες εγγραφές που γίνονται ταυτόχρονα.

- *Time Structured Merge Trees & Time Series Index*: Στην *Influxdb*, τα δεδομένα αποθηκεύονται μέσα σε ειδικά δομημένα αρχεία τα οποία ονομάζονται *TSM* (*Time Structured Merge Tree*) [14]. Όπως φαίνεται και στο σχήμα 2.9 τα δεδομένα σε ένα τέτοιο αρχείο ομαδοποιούνται σε *blocks* ανάλογα με το *Series Key* που ανήκουν. *Series Key* είναι ο συνδυασμός των στοιχείων *Measurement*, *Tags* και *Fields* στα οποία αναφερθήκαμε προηγουμένως. Επιπλέον, τα δεδομένα σε κάθε *block* είναι ταξινομημένα βάση της χρονοσφραγίδας τους. Αδιαμφισβήτητα, σημαντικό μέρος ενός *TSM* αρχείου, είναι και το ευρετήριο *TSI* (*Time Series Index*). Το ευρετήριο αποθηκεύει μεταδεδομένα σχετικά με το *block* στο οποίο βρίσκονται συγκεκριμένα *Series Keys*. Αναντίλεκτα, τα ευρετήρια *TSI* βοηθούν στην γρήγορη πρόσβαση δεδομένων και συνεπώς στην καλύτερη εξυπηρέτηση των εισερχόμενων *queries*.

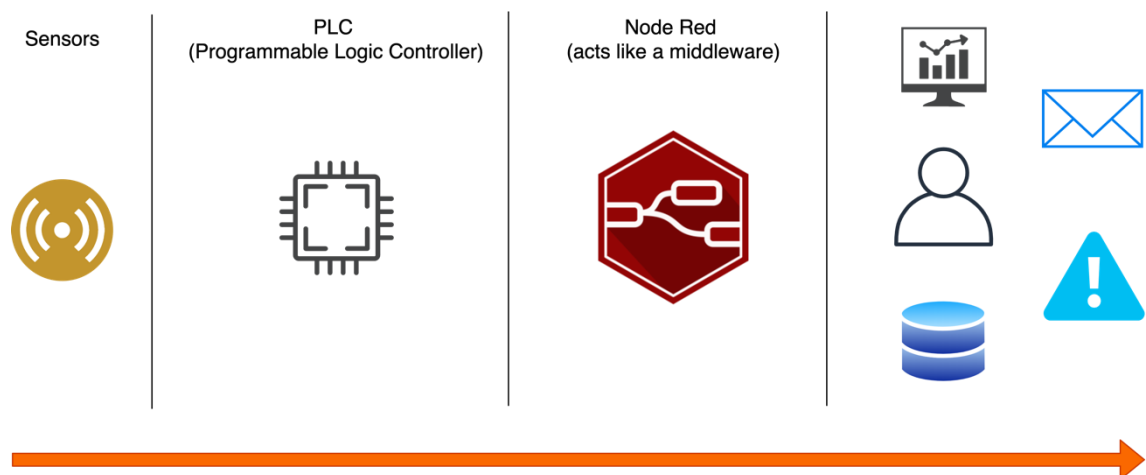


Σχήμα 2.8 Παράδειγμα αρχείου TSM

2.4 Διαχείριση Δεδομένων Διαδικτύου Αντικειμένων με το Ενδιάμεσο Λογισμικό Node-RED

Σε αυτή την υποενότητα, θα εξετάσουμε τον υφιστάμενο τρόπο με τον οποίο διαχειριζόμαστε *IoT* συσκευές σε μεγάλη κλίμακα.

Η αρχιτεκτονική που χρησιμοποιείται για βιομηχανικά *IoT* δεδομένα συγκροτείται από πολλές στρώσεις. Όπως φαίνεται και στο σχήμα 2.10 υπάρχουν τέσσερα κύρια επίπεδα. Αρχικά, έχουμε το πρώτο επίπεδο, στο οποίο βρίσκονται οι αισθητήρες που παράγουν δεδομένα. Τα δεδομένα που λαμβάνουμε από τους αισθητήρες μπορεί να είναι κυματομορφές, απόλυτες τιμές, τιμές σε κάποιο διάστημα ή να έχουν δυαδική τιμή. Στο δεύτερο επίπεδο, χρησιμοποιούμε κάποιο *PLC (Programmable Logic Controller)* για να ενορχηστρώσουμε και να ελέγξουμε τα δεδομένα που καταφθάνουν από το πρώτο επίπεδο. Παράδειγμα ενός τέτοιου υλισμικού μπορεί να είναι ένα *Raspberry Pi* ή ένα *Arduino* [16]. Το επόμενο κομμάτι του συστήματος είναι κάποιο ενδιάμεσο λογισμικό (*middleware*), το οποίο αναλαμβάνει την αυτοματοποίηση διάφορων διεργασιών, έτσι ώστε να οργανώσουμε, να αποθηκεύσουμε και να διανέμουμε τα δεδομένα αυτά. Ένα τέτοιο εργαλείο είναι και το *Node-Red* [17]. Τέλος, έχουμε τις υποδομές όπου αποθηκεύονται τα δεδομένα. Τέτοια συστήματα μπορεί να είναι βάσεις δεδομένων ένας *mail-server*, που ειδοποιά κάποιο χρήστη ή μια *front-end* εφαρμογή που παρουσιάζει στατιστικά στους χρήστες.



Σχήμα 2.9 Industrial IoT Architecture Example

2.5 Αποσύνθεση Μεγάλων Δεδομένων

Η εκτεταμένη χρήση των *IoT* συσκευών σε όλα τα φάσματα της ζωής μας, εγείρει έντονους προβληματισμούς σχετικά με την αποθήκευση και προσχώρηση της αυξανόμενης μάζας δεδομένων που συσσωρεύεται.

Το συγκεκριμένο θέμα πραγματεύονται και οι συγγραφείς της επιστημονικής δημοσίευσης «Decaying Telco Big Data with Data Postdiction» [18].

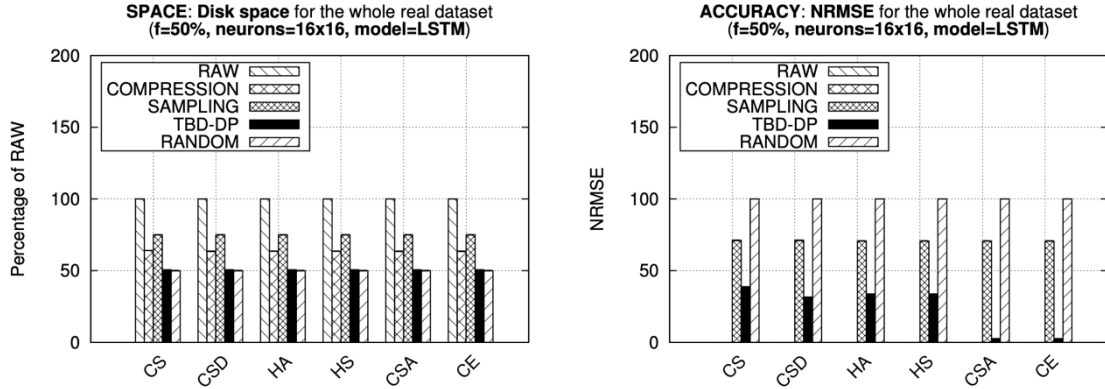
Εισαγωγικά, ας φέρουμε για παράδειγμα το σενάριο μιας μεγάλης εταιρείας τηλεπικοινωνιών, η οποία εξυπηρετεί εκατομμύρια ανθρώπους. Τα δεδομένα που παράγονται στους τηλεπικοινωνιακούς κόμβους (πχ ιστορικό κλήσεων), ανέρχονται σε 5TB ημερησίως [18] και η αποθήκευσή τους δεν είναι εύκολο εγχείρημα. Συγκεκριμένα, η συλλογή της πληροφορίας σε παραδοσιακές υποδομές σύννεφου (*cloud*), δεν είναι καλή ιδέα αφού η διαδικασία για να προσχωρήσουμε στα δεδομένα αυτά, σε μεταγενέστερο στάδιο, είναι υπολογιστικά ακριβή και ασύμφορη. Πέραν τούτου, η χρήση δημόσιων *cloud* όπως το AWS, είναι επικίνδυνη αφού τίθενται θέματα ασφάλειας και ιδιωτικότητας. Αναντίλεκτα, από την στιγμή που υπάρχει ένας τόσο μεγάλος όγκος δεδομένων η παροχή ενεργών αντίγραφων ασφαλείας είναι σχεδόν αδύνατη. Επιπλέον, οι τηλεπικοινωνιακοί πάροχοι μαζεύουν περισσότερα δεδομένα απ' ό,τι χρειάζονται, με αποτέλεσμα ένα μεγάλο μέρος της πληροφορίας αυτής να μην είναι χρήσιμο.

Μια προσέγγιση στην οποία στηρίζεται η βιομηχανία εδώ και αρκετό καιρό, είναι η συμπίεση των δεδομένων. Με αυτό τον τρόπο εξοικονομείται αποθηκευτικός χώρος ενώ ταυτόχρονα διατηρούμε τα δεδομένα ακέραια. Παρόλα αυτά, για να προσχωρήσουμε στα δεδομένα χρειάζεται κάθε φορά να τα αποσυμπιέσουμε και να τα συμπιέσουμε, πράγμα που δεν είναι πάντα δόκιμο. Αναλυτικότερα, από τα πειράματα που έχουν διεξάγει οι συγγραφείς του ακαδημαϊκού άρθρου, αναφέρεται ότι η τεχνική της συμπίεσης κατάφερε να μειώσει τον όγκο των δεδομένων περίπου στο 60%.

Ένας άλλος τρόπος που λήφθηκε υπόψη στα πειράματα που διενεργήθηκαν, είναι η λήψη τυχαίων δειγμάτων με ομοιόμορφη κατανομή. Η μέθοδος αυτή είναι αρκετά αποτελεσματική ως προς την συρρίκνωση του όγκου δεδομένων, αφού διατηρείται μόνο το 50% τους. Παρόλα αυτά, με την μέθοδο αυτή χάνεται αρκετή πληροφορία και στην προσπάθεια ανάγνωσης κάποιας τιμής για μια δεδομένη χρονική στιγμή στο παρελθόν δεν έχουμε και τόσο καλά αποτελέσματα. Αυτό φαίνεται καλύτερα και στο σχήμα 2.11

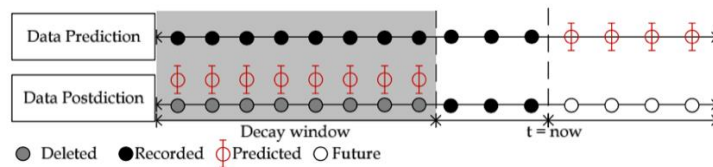
Σε συνέχεια της προσπάθειας αυτής, έγινε η επιλογή δείγματος από την παραγόμενη ροή δεδομένων για κάθε δευτερόλεπτο που περνούσε. Σαφώς, με αυτό τον τρόπο διατηρείται περισσότερη πληροφορία. Σε αυτή την μέθοδο ο συνολικός όγκος δεδομένων ανέρχεται στο 75% του αρχικού, ενώ παράλληλα σύμφωνα με τα πειράματα στο σχήμα 2.11, το

NRMSE για την πρόβλεψη δεδομένων είναι μικρότερο σε σχέση με αυτό της ομοιόμορφης κατανομής.



Σχήμα 2.10 Αποτίμηση TBD-DP σε σχέση με τον χώρο και την ακρίβεια προβλέψεων

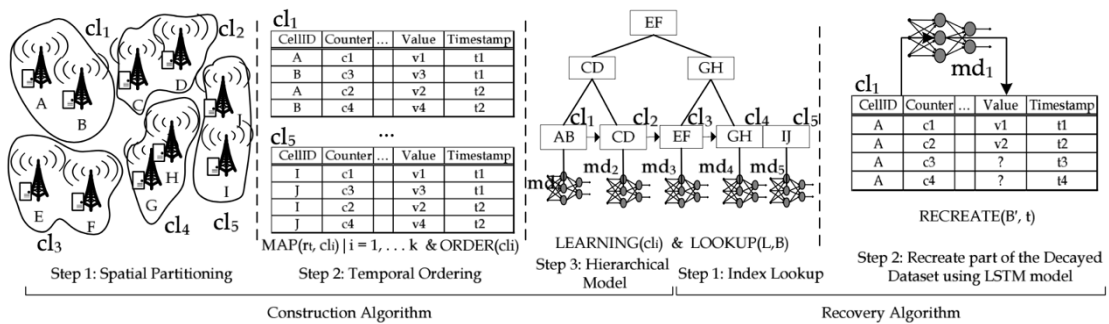
Τέλος, η καινοτόμα ιδέα που προτείνεται στο άρθρο, αφορά την χρήση μοντέλων *Machine learning* για την καλύτερη πρόβλεψη τιμών που έχουν παραχθεί στο παρελθόν, από τους τηλεπικοινωνιακούς κόμβους. Συγκεκριμένα, η μέθοδος αυτή ονομάζεται «Postdiction» και αφορά την χρήση *LSTM* μοντέλων για την εκπαίδευση ενός νευρωνικού δικτύου. Ο όρος «Postdiction» [18] περιγράφεται καλύτερα στο σχήμα 2.12. Οι νευρώνες *LSTM* [19] είναι αναδρομικοί και έχουν την ιδιότητα να κρατούν πληροφορίες από το παρελθόν στην μνήμη τους. Μπορούμε να ελέγξουμε το ποσοστό της πληροφορίας που κρατούν, ορίζοντας έναν «decay factor», ο οποίος χρησιμοποιείται κατά την *offline* φάση εκπαίδευσης του νευρωνικού δικτύου.



Σχήμα 2.11 Data Postdiction

Σε αυτό το σημείο θα δούμε σε εννοιολογικό επίπεδο, τον αλγόριθμο κατασκευής του μοντέλου και τον αλγόριθμο ανάκτησης δεδομένων όπως φαίνεται στην εικόνα 2.12.

- *Αλγόριθμος κατασκευής*
 1. Αυτό το στάδιο είναι προπαρασκευαστικό. Εδώ καθορίζουμε τον συντελεστή «f» ο οποίος υποδεικνύει το ποσοστό των δεδομένων που θα ξεχνά το νευρωνικό δίκτυο κατά την εκπαίδευσή του.
 2. Το δεύτερο στάδιο ονομάζεται «Spatial Partitioning». Εδώ γίνεται διαχωρισμός και ομαδοποίηση των τηλεπικοινωνιακών κόμβων βάση της χωρικής τους τοποθέτησης.
 3. Τα τηλεπικοινωνιακά δεδομένα ταξινομούνται βάση της χρονοσφραγίδας τους.
 4. Σε αυτό το σημείο τα δεδομένα που σχηματίστηκαν στο προηγούμενο βήμα, εκπαιδεύουν τους νευρώνες *LSTM* και παράγουν μοντέλα, τα οποία τοποθετούνται ιεραρχικά σε ένα *DP-Tree* [18] για εύκολη και γρήγορη πρόσβαση σε αυτά.
- *Αλγόριθμος ανάκτησης*
 1. Για να ανακτήσουμε ένα συγκεκριμένο μοντέλο ανατρέχουμε στο *DP-Tree* το οποίο δημιουργήθηκε στο 3^ο βήμα του κατασκευαστικού αλγόριθμου.
 2. Τέλος, οι πληροφορίες που αναζητούμε ανακατασκευάζονται με την χρήση του *LSTM* μοντέλου που ανακτήθηκε στο 1^ο βήμα.

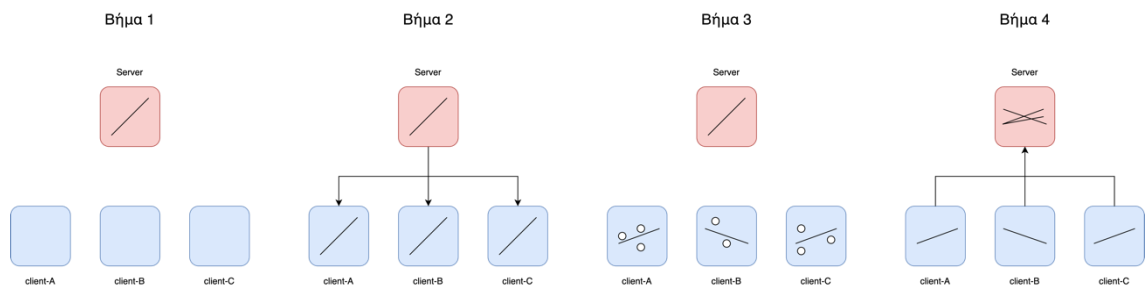


Σχήμα 2.12 Τα εννοιολογικά βήματα για τον αλγόριθμο κατασκευής και ανάκτησης

2.6 Τεχνική Ομοσπονδιακής Μάθησης

Το *Federated Learning* [20] [21] είναι μια κατανεμημένη τεχνική μηχανικής μάθησης η οποία εξετάστηκε στα πλαίσια της παρούσας διπλωματικής εργασίας και μπορεί να χρησιμοποιηθεί σε μελλοντική δουλειά.

Αναλυτικότερα, το *Federated Learning* είναι μια πρόταση της επιστημονικής ομάδας της Google, όπου η εκπαίδευση διάφορων μοντέλων γίνεται στα άκρα του δικτύου, πάνω στις συσκευές οι οποίες παράγουν τα δεδομένα. Η κύρια ιδέα πίσω από αυτή την προσέγγιση, είναι ότι τα δεδομένα δεν μεταδίδονται από την συσκευή σε κάποιο server για να εκπαιδευτεί το μοντέλο. Με αυτό τον τρόπο αυτό διατηρείται η ιδιωτικότητα και η ασφάλεια των δεδομένων ως προς τρίτους. Πιο κάτω θα δούμε σε συντομία την αρχιτεκτονική *Federated Learning*.



Σχήμα 2.13 Το μοντέλο του *Federated Learning*

1. Ο server επιλέγει κάποιο στατιστικό ή ML μοντέλο που θέλουμε να εκπαιδύσουμε.
2. Το μοντέλο που επιλέγηκε στο πρώτο βήμα διανέμεται στους *Clients*, όπου παράγονται τα δεδομένα.
3. Κάθε *Client* εκπαιδεύει το μοντέλο τοπικά με τα δεδομένα που έχει παράξει.
4. Αφού τελειώσει η εκπαίδευση των μοντέλων στους *Clients* τα μοντέλα προωθούνται στον server, ο οποίος αναλαμβάνει την σύνοψή τους και τη δημιουργία ενός καινούριου μοντέλου το οποίο διανέμεται σε όλους τους *Clients*. Η διαδικασία επαναλαμβάνεται με νέα δεδομένα που παράγονται στους *Clients*.

Κεφάλαιο 3

Τεχνολογικό Υπόβαθρο

3.1 Δίκτυα Ομότιμων Κόμβων.....	29
3.2 Κρυπτογραφία.....	29
3.2.1 Κρυπτογραφία Δημόσιου Κλειδιού	30
3.2.2 Συναρτήσεις Κατακερματισμού	30
3.2.3 Ψηφιακές Υπογραφές	31
3.3 Κωδικοποίηση Base64.....	32
3.4 Hyperledger Fabric.....	32
3.3.1 Δοσοληψίες.....	34
3.3.2 Το μοντέλο του Hyperledger Fabric	34
3.3.3 Αλυσίδα Συστοιχιών	36
3.3.4 Ομότιμοι Κόμβοι	37
3.3.5 Υπηρεσία Σειριοποίησης	39
3.3.6 Έξυπνα Συμβόλαια	41
3.3.7 Καθολικό.....	42

Σε αυτό το κεφάλαιο θα δούμε βασικές έννοιες οι οποίες σχετίζονται με το τεχνολογικό υπόβαθρο το οποίο πρέπει να είναι κανείς οικείος για να αντιληφθεί τον τρόπο λειτουργίας της *Triabase*, όπως αυτή θα περιγράφει στο κεφάλαιο 4. Αναλυτικότερα, θα δούμε γενικότερους ορισμούς όπως τα *Peer-to-Peer* δίκτυα, ενώ σε μεταγενέστερο στάδιο θα εξετάσουμε ενδελεχώς τον τρόπο λειτουργίας του *Hyperledger Fabric*, ενός *Blockchain framework*, το οποίο χρησιμοποιήσαμε για την υλοποίηση του συστήματός μας.

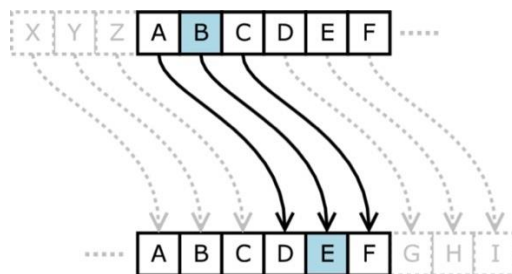
3.1 Δίκτυα Ομότιμων Κόμβων

Με τον όρο δίκτυο ομότιμων κόμβων [22] [23] δίκτυο αναφερόμαστε σε μια κατανεμημένη αρχιτεκτονική δικτύου η οποία αποτελείται από δύο ή περισσότερες συσκευές. Όλα τα μέλη του δικτύου επικοινωνούν μεταξύ τους χωρίς την ύπαρξη κάποιου κεντρικού διακομιστή. Έτσι, για να επιτευχθεί επικοινωνία στο δίκτυο, κάθε συσκευή έχει τον ρόλο του *server* και *Client* ταυτόχρονα. Βασική προϋπόθεση για να μπορέσει μια συσκευή να συμμετάσχει σε ένα τέτοιο δίκτυο, είναι η χρήση P2P λογισμικού προς. Γνωστό παραδείγματα P2P δικτύου είναι το *BitTorrent* [24].

3.2 Κρυπτογραφία

Στο άκουσμα προς λέξης κρυπτογραφία το μυαλό προς έρχονται εικόνες όπου άνθρωποι προσπαθούν να σπάσουν τον κώδικα πίσω από ένα κρυπτογραφημένο μήνυμα ή την ασφαλή πρόσβαση στο *e-mail* προς ή σε μία ιστοσελίδα [25]. Είναι λοιπόν οφθαλμοφανές ότι η κρυπτογραφία τείνει να είναι στενά συνδεδεμένη με την έννοια προς σύγχρονης ασφαλής επικοινωνίας.

Παρόλα αυτά, η επιστήμη αυτή έχει προς ρίζες προς προς αρχαίους πολιτισμούς. Για παράδειγμα, στην αρχαία Ρώμη χρησιμοποιούσαν αυτό που προς μέρες προς ονομάζουμε «Caesar Cipher» [26], έναν απλό μηχανισμό που χρησιμοποιούσαν οι άνθρωποι για να μεταβιβάζουν με ασφάλεια μηνύματα πολέμου κλπ. Στο σχήμα 3.1 βλέπουμε πιο αναλυτικά παράδειγμα χρήσης του «Caesar Cipher».



Σχήμα 3.1 Παράδειγμα Caesar Cipher

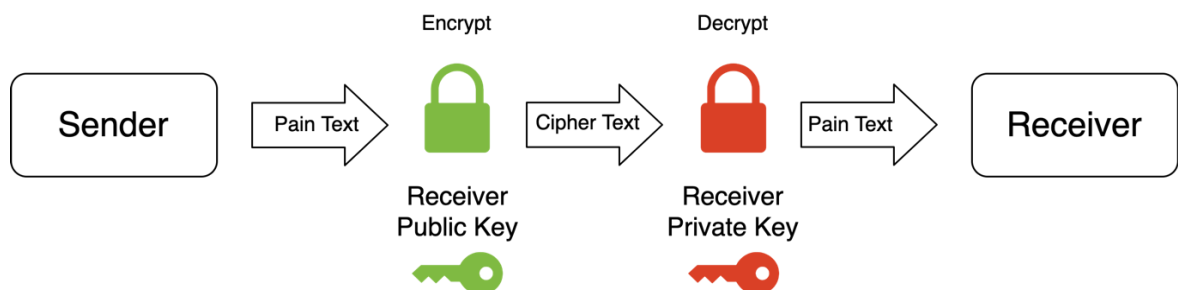
Τετριμμένα ο πιο διαδεδομένος τρόπος κρυπτογραφίας είναι με την χρήση συμμετρικού κλειδιού [25]. Παρόλα αυτά, η ανταλλαγή του κλειδιού σε ένα ανοικτό περιβάλλον εγκυμονεί κινδύνους.

3.2.1 Κρυπτογραφία Δημόσιου Κλειδιού

Στην ψηφιακή εποχή την οποία ζούμε, η χρήση του πρότυπου «Public Key Cryptography» [25] [27] είναι αναγκαία για την ύπαρξη εμπιστευτικότητας, ακεραιότητας και αυθεντικότητας [28] μεταξύ διαφόρων μετόχων σε ένα δίκτυο.

Αναλυτικότερα, στο πρότυπο «Public Key Cryptography» χρησιμοποιούμε ένα ζευγάρι από κλειδιά, τα οποία κατασκευάζουμε με την χρήση μαθηματικών εξισώσεων [29]. Το ένα κλειδί είναι δημόσιο προς όλους, ενώ το δεύτερο κλειδί είναι ιδιωτικό. Δηλαδή, μόνο ο ιδιοκτήτης του έχει πρόσβαση σε αυτό.

Στο σχήμα 3.2 βλέπουμε τον μηχανισμό «Public Key Cryptography» σε λειτουργία.



Σχήμα 3.2 Παράδειγμα χρήσης Public Key Cryptography

3.2.2 Συναρτήσεις Κατακερματισμού

Μια συνάρτηση κατακερματισμού [30], χρησιμοποιείται για την αντιστοίχιση μιας αυθαίρετης ακολουθίας δεδομένων σε μια μοναδική τιμή συγκεκριμένου μεγέθους. Το παραγόμενο αποτέλεσμα ονομάζεται *Hash* ή *Digest*. Μια συνάρτηση κατακερματισμού πρέπει να ικανοποιεί τους ακόλουθους τρεις περιορισμούς:

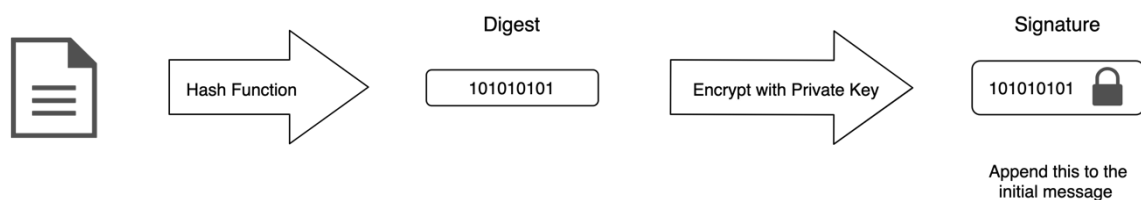
- *Preimage resistance* [30] : Δεδομένης συνάρτησης κατακερματισμού ϕ και *Digest* $\phi(\chi)$, πρέπει να είναι αδύνατον να προσδιορίσουμε την αρχική ακολουθία δεδομένων χ .
- *Second Preimage resistance* [30]: Δεδομένου μηνύματος μ και του αντίστοιχου *Digest* $\phi(\mu)$ πρέπει να είναι δύσκολο να βρούμε ένα μήνυμα λ τέτοιο ώστε $\phi(\mu) = \phi(\lambda)$.
- *Collision resistance* [30]: Πρέπει να είναι αδύνατη η εύρεση δύο διαφορετικών μηνυμάτων μ και λ τέτοια ώστε να έχουν το ίδιο *Digest*.

3.2.3 Ψηφιακές Υπογραφές

Ένα ακόμη εργαλείο που χρησιμοποιούμε στην κρυπτογραφία είναι οι ψηφιακές υπογραφές [31]. Οι ψηφιακές υπογραφές λειτουργούν όμοια με τις χειρόγραφες υπογραφές. Δηλαδή παρέχουν έναν μηχανισμό ο οποίος εγγυείται ότι το μήνυμα που έχει σταλεί από μια οντότητα σε μια άλλη είναι αυθεντικό και δεν έχει παραποιηθεί από κάποια άλλη κακόβουλη οντότητα.

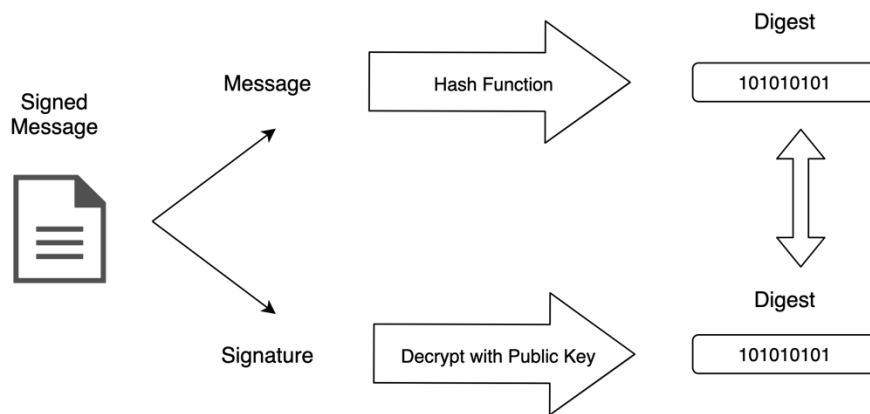
Η χρήση των ψηφιακών υπογραφών γίνεται με τα ακόλουθα δύο βήματα:

1. *Υπογραφή*: Αρχικά το μήνυμα που θέλουμε να υπογράψουμε δίνονται ως είσοδος σε μια συνάρτηση κατακερματισμού. Αφού πάρουμε το παραγόμενο *Digest* το κρυπτογραφούμε με το Private Key μας. Το αποτέλεσμα αυτής της διαδικασίας είναι μια ψηφιακή υπογραφή και προστίθεται στο τέλος του μηνύματος που θέλουμε να στείλουμε.



Σχήμα 3.3 Διαδικασία Digital Signing

2. *Επαλήθευση υπογραφής*: Ο παραλήπτης του μηνύματος για να επαληθεύσει το γνησιότητα του, πρέπει να ακολουθήσει την διαδικασία επαλήθευσης. Ειδικότερα, ο παραλήπτης εισάγει στην αντίστοιχη συνάρτηση κατακερματισμού το μήνυμα και παράγει ένα *Digest*. Παράλληλα, χρησιμοποιεί το Public Key του αποστολέα για να αποκρυπτογραφήσει την υπογραφή. Αν το αποκρυπτογραφημένο *Digest* είναι το ίδιο με αυτό που έχει παράξει η συνάρτηση κατακερματισμού, τότε το μήνυμα είναι γνήσιο και δεν έχει παρεμβεί κάποια κακόβουλη οντότητα σε αυτό.



Σχήμα 3.4 Διαδικασία *Digital Signature Verification*

3.3 Κωδικοποίηση *Base64*

Base64 είναι ένας τρόπος που χρησιμοποιούμε για να κωδικοποιήσουμε δυαδικούς μη εκτυπώσιμους χαρακτήρες σε *ASCII* μορφή. Σκοπός του *Base64* είναι μετατροπή δυαδικών αρχείων, όπως είναι οι φωτογραφίες, σε μια συμβολοσειρά έτσι ώστε να μπορούμε να διανέμουμε τα δεδομένα αυτά μέσω σε ένα δίκτυο, με μεγαλύτερη ευκολία.

3.4 *Hyperledger Fabric*

Το *Hyperledger Fabric* [32] [33] είναι ένα εργαλείο ανοικτού κώδικα το οποίο επιτρέπει την ανάπτυξη κατανεμημένων καθολικών (*Distributed Ledgers*). Πιο αναλυτικά είναι σχεδιασμένο για επαγγελματική χρήση μεταξύ οργανισμών και για αυτό τον λόγο εντάσσεται στην κατηγορία των ιδιωτικών αλυσίδων συστοιχίων μια δικαιώματα (*Private Permissioned Blockchains*). Πέραν τούτου η αρχιτεκτονική του *Hyperledger*

Fabric είναι ιδιαίτερα ενδιαφέρουσα και διαφορετική σε σχέση με τις υπόλοιπες συμβατικές τεχνολογίες αλυσίδων συστοιχιών (*Blockchain*).

Το *Hyperledger Fabric*, παρέχει την δυνατότητα στους προγραμματιστές να τροποποιήσουν το σύστημα, αλλάζοντας διαφορετικές παραμέτρους, ανάλογα με τις ανάγκες της εφαρμογής που υλοποιούν. Συγκεκριμένα η εξαιρετικά αρθρωτή αρχιτεκτονική του *Fabric* επιτρέπει την χρήση του σε ένα ευρύ φάσμα από περιπτώσεις συμπεριλαμβανομένων του τραπεζικού τομέα, του τομέα υγείας, του τομέα ανάπτυξης και διαχείρισης ανθρώπινου δυναμικού και σε διάφορα *supply chains*.

Επιπλέον, μια από τις σημαντικότερες λειτουργίες που παρέχει το *Fabric* είναι η χρήση έξυπνων συμβολαίων (*smart contracts* [34] ή αλλιώς *chaincode*). Το *chaincode* μπορεί να γραφτεί σε διάφορες γλώσσες προγραμματισμού όπως η Go, Java και NodeJS. Αυτό το ανταγωνιστικό πλεονέκτημα επιτρέπει στις εταιρείες να χρησιμοποιήσουν εύκολα την τεχνολογία του *Fabric*, χωρίς το προσωπικό να χρειάζεται εκπαίδευση σε καινούριες γλώσσες προγραμματισμού όπως για παράδειγμα η *Solidity* [35]. (*Solidity* είναι μια γλώσσα προγραμματισμού που δημιουργήθηκε ειδικά για την συγγραφή *smart contracts* στην πλατφόρμα του *Ethereum*).

Επίσης, ένα από τα σημαντικότερα χαρακτηριστικά του *Hyperledger Fabric* είναι ότι ανήκει στην κατηγορία *Private Permissioned Blockchains*. Σε ένα *Public Blockchain* όπως το *Bitcoin*, οι συμμετέχοντες είναι ανώνυμοι και δεν γνωρίζουν ο ένας τον άλλο, με αποτέλεσμα να μην υπάρχει εμπιστοσύνη. Αντίθετα, σε ένα ιδιωτικό δίκτυο οι συμμετέχοντες γνωρίζονται μεταξύ τους και υπάρχει ένα περιβάλλον αυξημένης εμπιστοσύνης, το οποίο πλαισιώνεται από μοντέλα δίκαιης διακυβέρνησης και προκαθορισμένους κανόνες. Ακόμη, το *Fabric* παρέχει την δυνατότητα για δικαιώματα. Αυτό σημαίνει πως σε ένα δίκτυο υπάρχουν χρήστες που μπορούν να κάνουν ορισμένες ενέργειες, όπως διάβασμα και γράψιμο στο *Ledger*, ενώ άλλοι μπορούν μόνο να διαβάσουν από αυτό. Η λειτουργία αυτή, συντείνει στην ασφαλέστερη, αποτελεσματικότερη διαχείριση και ομαλή λειτουργία του δικτύου.

Ένα από τα πιο σημαντικά χαρακτηριστικά της πλατφόρμας είναι η υποστήριξη διαφορετικών *consensus* αλγορίθμων, ανάλογα με το επίπεδο εμπιστοσύνης που υπάρχει

μεταξύ των συμμετεχόντων. Ειδικότερα, υπάρχουν δύο μοντέλα αλγορίθμων ομοφωνίας (*consensus*) που μπορούν να χρησιμοποιηθούν.

- *Crash Fault Tolerant*: Χρησιμοποιείται σε περιπτώσεις όπου οι κόμβοι του δικτύου είναι πιο πιθανόν να καταρρεύσουν. Παράδειγμα ενός τέτοιου αλγόριθμου είναι το *RAFT* [36].
- *Byzantine Fault Tolerant*: Χρησιμοποιείται σε περιπτώσεις όπου υπάρχει αυξημένη πιθανότητα ύπαρξης κακεντρεχούς συμμετέχοντα στο δίκτυο. Παράδειγμα ενός τέτοιου αλγόριθμου είναι το *PBFT* [7].

Συμπληρωματικά, οι αλγόριθμοι αυτού του τύπου δεν απαιτούν την ύπαρξη κάποιου κρυπτονομίσματος. Με τον τρόπο αυτό εξασφαλίζουμε την ασφάλεια του δικτύου ενώ ταυτόχρονα τα έξοδα ανέγερσης του είναι μικρά. Σε αντίθεση, άλλα *Blockchains* όπως το *Bitcoin* χρησιμοποιούν κοστοβόρους *consensus* αλγόριθμους (*Proof of Work* [7]) οι οποίοι χρειάζονται εξεζητημένους υπολογιστικούς πόρους.

Ο συνδυασμός των προαναφερθέντων χαρακτηριστικών καθιστά το *Hyperledger Fabric* ως ένα από τα καλύτερα *Blockchain* εργαλεία όσον αφορά την επεξεργασία δοσοληψιών (*transaction processing*) και καθυστέρηση επικύρωσης δοσοληψιών (*transaction confirmation Latency*).

3.3.1 Δοσοληψίες

Δοσοληψία, συναλλαγή ή *transaction* ονομάζουμε μια ατομική ακολουθία από αναγνώσεις και εγγραφές σε μια βάση δεδομένων [15]. Την αντίστοιχη έννοια συναντούμε και στον κόσμο του *Blockchain* όπου διάφορα *transactions* καταγράφονται σε ένα κατανεμημένο *Ledger*.

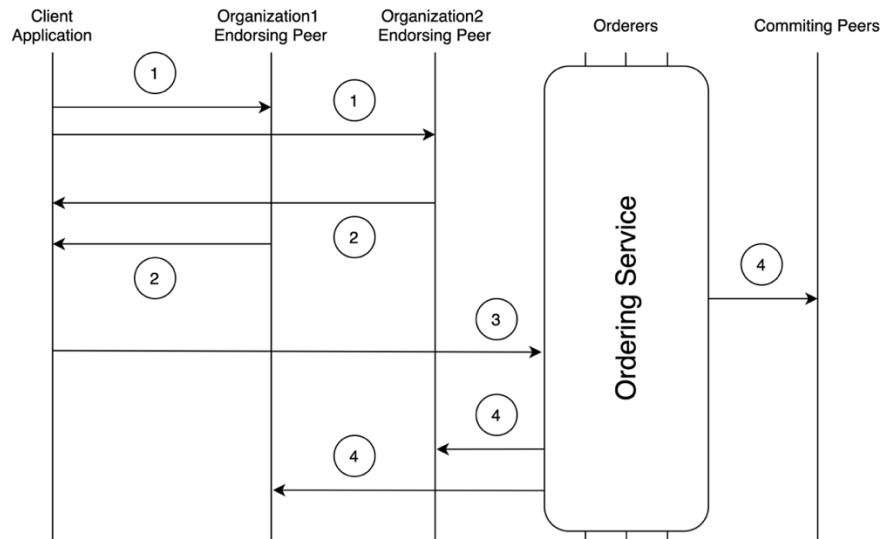
3.3.2 Το μοντέλο του *Hyperledger Fabric*

Το *Hyperledger Fabric* εισάγει μια καινούργια αρχιτεκτονική όσον αφορά την διαχείριση *transactions*. Η μεθοδολογία αυτή ονομάζεται *execute-order-validate* [32] και

εξασφαλίζει την ευελιξία, κλιμακωσιμότητα και απόδοση. Τα ακόλουθα τρία βήματα περιγράφουν την διαδικασία αυτή με μεγαλύτερη λεπτομέρεια.

- *Execute*: Το *transaction* προωθείται ως πρόταση από το *Client* σε ένα *Peer*. Το *Peer* εκτελεί το *transaction* και ελέγχει για την ορθότητα του. Αν είναι ορθό, το *transaction* γίνεται endorse από το συγκεκριμένο *Peer*.
- *Order*: Τα *transactions* μπαίνουν σε μια σειρά από τους *Orderers* οι οποίοι χρησιμοποιούν κάποιο *consensus* πρωτόκολλο.
- *Validate*: τα *transactions* γίνονται validate βάση του endorsement και άλλων κανόνων. Ακολούθως γίνεται το τελικό *commit* στο *Ledger*.

Στο πιο κάτω σχήμα βλέπουμε πιο αναλυτικά τον κύκλο ζωής κάποιου *transaction* από την στιγμή που γεννιέται μέχρι το τελικό *commit* στο κάθε *Peer* του δικτύου.



Σχήμα 3.5 Hyperledger Fabric Transaction Flow

- *Στάδιο 1* : Αρχικά ο *Client* δημιουργεί ένα καινούργιο *transaction Proposal* το οποίο προωθεί στα *Endorsing Peers*. $Tx\ Proposal = \langle ClientID, ChaincodeID, txPayload, Timestamp, ClientSignature \rangle$.
- *Στάδιο 2* : Αφού τα *Endorsing Peers* παραλάβουν το *Transaction Proposal* επαληθεύουν ότι είναι στην σωστή μορφή, ελέγχουν ότι η συγκεκριμένη πρόταση δεν έχει υποβληθεί ξανά στο παρελθόν και ότι το signature του *Client* είναι έγκυρο. Τέλος, γίνεται έλεγχος των endorsing policies ούτως ώστε να βεβαιωθούμε ότι ο χρήστης που υπέβαλε το *transaction* αυτό έχει το δικαίωμα

να επέμβει στο συγκεκριμένο κανάλι και *chaincode*. Στην συνέχεια, γίνεται εκτέλεση/προσομοίωση του *transaction* έναντι του υφιστάμενου *World State database* [33] και παράγεται ένα μήνυμα που περιέχει τα read/write sets (ζευγάρια κλειδιών/τιμών των αντικειμένων (objects) που θέλουμε να δημιουργήσουμε ή να ενημερώσουμε) του *Transaction*. Το μήνυμα αυτό σε συνδυασμό με signature του *Endorsing Peer* επιστρέφονται ως απάντηση στον *Client*. Response = < Read/Write Set, Endorsing Peer Signature>

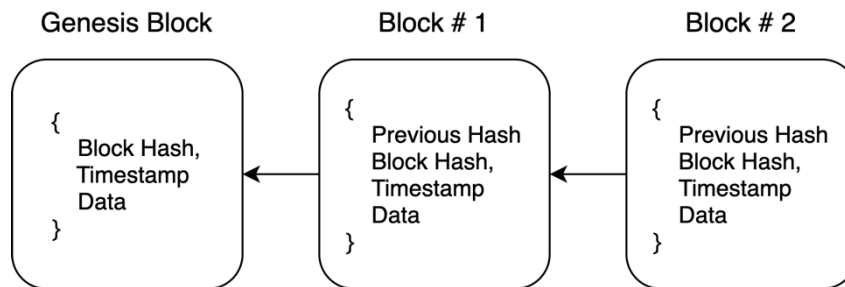
- *Στάδιο 3* : Το application παραλαμβάνει τα μηνύματα από όλα τα *Endorsing Peers* και προχωρεί στην επαλήθευση των signatures. Επίσης, γίνεται έλεγχος αν όλες οι απαντήσεις είναι ίδιες. Αφού ικανοποιηθούν οι πιο πάνω απαιτήσεις το *Transaction* είναι έτοιμο για αναμετάδοση στους *Orderers*.
- Το *Ordering Service* παραλαμβάνει όλα τα εισερχόμενα *transactions* από διάφορους *Clients* και τα ενσωματώνουν σε ένα *block* ανάλογα με την χρονοσφραγίδα τους.
- *Στάδιο 4* : Αφού το *block* είναι έτοιμο αναμεταδίδεται σε όλα τα *Peers* στο κανάλι με απώτερο σκοπό να γίνουν *commit* τοπικά σε κάθε ένα από αυτά. Πριν το τελικό βήμα του *commit* όλα τα *transactions* επικυρώνονται για να σιγουρευτούμε ότι δεν έχει προκύψει κάποια αλλαγή στο ενδιαμέσο. Σημαντικό είναι να αναφέρουμε ότι αφού ενημερωθεί το *Ledger*, ο *Client* ενημερώνεται με την χρήση του μηχανισμού event emit.

3.3.3 Αλυσίδα Συστοιχιών

Σε αυτή την υποενότητα θα δούμε περιληπτικά σε εννοιολογικό επίπεδο πως το *Hyperledger Fabric* επιτρέπει σε διάφορους οργανισμούς να διαμορφώσουν ένα δίκτυο *Blockchain*.

Γενικά μπορούμε να περιγράψουμε το *Blockchain* ως μια κατανεμημένη δομή δεδομένων η οποία απαρτίζεται από πολλά *Block*. Κάθε *block* περιλαμβάνει ένα σύνολο από *transactions* τις οποίες πραγματοποιούν όσοι συμμετέχουν στο δίκτυο. Ακόμη, τα *block* είναι συνδεδεμένα μεταξύ τους χρησιμοποιώντας κρυπτογραφία, αφού κάθε ένα από αυτά περιλαμβάνει το *Hash* που έχει ένα προηγούμενο *block*. Έτσι διασφαλίζεται η ακεραιότητα και η αξιοπιστία των δεδομένων που βρίσκονται μέσα σε αυτά. Στο ποιο

κάτω σχήμα βλέπουμε την αναπαράσταση μιας τέτοιας δομής δεδομένων. Αν τα δεδομένα σε ένα από αυτά τα *block* αλλάξει αλλάζουν τα *Hash* των *Blocks* με αποτέλεσμα να σπάει η αλυσίδα.



Σχήμα 3.6 Παράδειγμα Blockchain με 3 Block

Το *Blockchain* του *Hyperledger Fabric* μπορεί να φιλοξενεί διάφορα είδη εφαρμογών. Χρησιμοποιώντας *smart contracts* μπορούμε να φτιάξουμε διάφορες εφαρμογές οι οποίες δημιουργούν *transactions* τα οποία σε μεταγενέστερο στάδιο διανέμονται σε κάθε ομότιμο κόμβο του δικτύου, όπου καταγράφονται αμετάβλητα στο αντίστοιχο *Ledger*.

3.3.4 Ομότιμοι Κόμβοι

Ένα δίκτυο *Blockchain* αποτελείται από ένα σύνολο κόμβων ή αλλιώς *Peers*. Οι κόμβοι αυτοί αποτελούν τα θεμέλια ενός τέτοιου συστήματος, αφού διατηρούν αντίγραφα του *Ledger* και φιλοξενούν τα *smart contracts*.

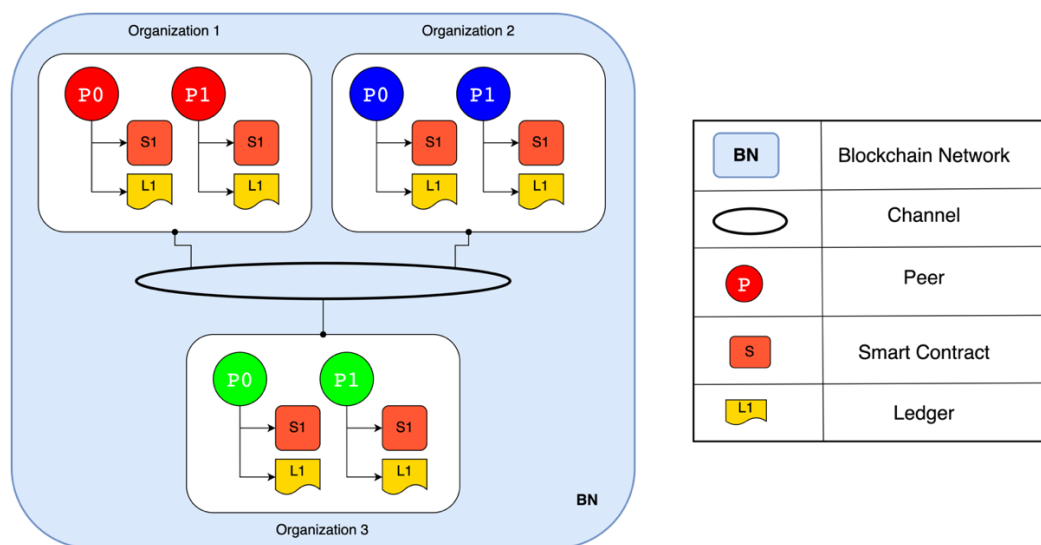
Στο *Hyperledger Fabric* τα *Peers* βρίσκονται υπό την ιδιοκτησία και αιγίδα των οργανισμών που συμμετέχουν στο συγκεκριμένο δίκτυο. Κάθε οργανισμός μπορεί να έχει ένα ή περισσότερα *Peers* ανάλογα με τον σχεδιασμό και την τοπολογία του δικτύου. Αναλυτικότερα, οι οργανισμοί μπορούν να δημιουργήσουν, ενεργοποιήσουν, απενεργοποιήσουν, διαμορφώσουν και διαγράψουν *Peers* ανάλογα με τις ανάγκες τους. Επιπλέον οι διαχειριστές και οι χρήστες του δικτύου μπορούν να αλληλοεπιδρούν με τα *Peers* μέσω μιας πληθώρας από *APIs* τα οποία παρέχει το *Hyperledger Fabric*.

Κάθε *Peer* διατηρεί ένα αντίγραφο του *Ledger*. Αυτό γίνεται ούτως ώστε να εξασφαλίσουμε την εμπιστοσύνη και διαφάνεια μεταξύ των συναλλαγών που λαμβάνουν μέρος στο δίκτυο. Ειδικότερα, σε σενάρια όπου υπάρχουν διενέξεις μεταξύ των μελών του δικτύου μπορούμε να αντιπαραβάλουμε τα *Ledgers* των *Peers* έτσι ώστε να δούμε

που βρίσκεται το πρόβλημα. Επιπλέον, στο *Ledger* μπορούμε να αποθηκεύσουμε το ιστορικό των *transactions* για ένα συγκεκριμένο object.

Εκτός από το *Ledger*, σε κάθε *Peer* μπορούμε να βρούμε εγκατεστημένα τα διάφορα *smart contracts* του δικτύου. Στα *smart contracts* οι προγραμματιστές μπορούν να υλοποιήσουν την επιχειρηματική λογική της εφαρμογής. Όταν ένα νέο *transaction* καταφθάσει στο *Peer* το αντίστοιχο *smart contract* ενεργοποιείται και γίνονται οι απαιτούμενες εγγραφές ή αναγνώσεις από το *Ledger*. Περισσότερες πληροφορίες για την έννοια του *smart contract* θα δούμε σε μεταγενέστερη υποενότητα.

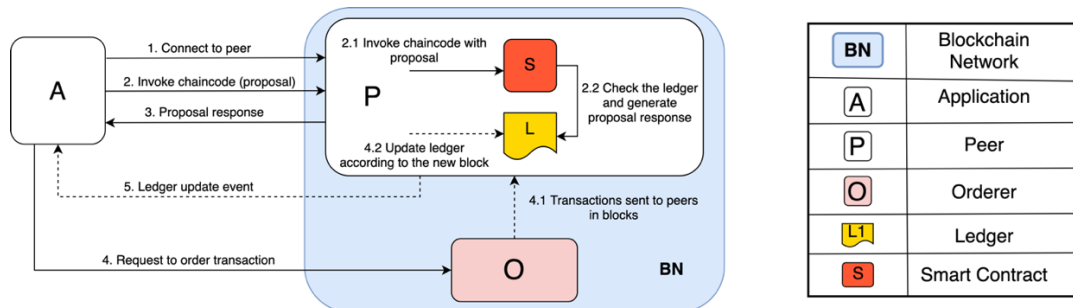
Στο σχήμα 3.7 απεικονίζεται ένα δίκτυο με τρεις οργανισμούς οι οποίοι διατηρούν δύο *Peers* αντίστοιχα. Επίσης, μπορούμε να δούμε ότι σε κάθε *Peer* υπάρχει *Ledger*, και το αντίστοιχο *smart contract*. Τα *Peers* επικοινωνούν μεταξύ τους κάνοντας εγγραφή σε ένα κοινό κανάλι.



Σχήμα 3.7 Παράδειγμα Δικτύου

Στο σχήμα 3.8. Βλέπουμε αναλυτικότερα τις κύριες διεργασίες ενός *Peer* που συμμετέχει στο *Blockchain*. Όπως έχει αναφερθεί στην υποενότητα «Το μοντέλο του *Hyperledger Fabric*» ένα *Peer* είναι ίσως το πιο σημαντικό μέρος στην εκτέλεση ενός *transaction*. Όπως φαίνεται και στο σχήμα, ένα *Peer* επικοινωνεί με το χρήστη διαμέσου κάποιας

εφαρμογής ενώ ταυτόχρονα επικοινωνεί με τον *Orderer* για να παραλάβει και να προωθήσει συναλλαγές από και προς όλα τα άλλα *Peers* του δικτύου.



Σχήμα 3.8 Αλληλεπίδραση εφαρμογής *Peer*

Τέλος, αξίζει να σημειώσουμε ότι σε κάθε *Peer* τρέχει μια βάση δεδομένων η οποία σκοπό έχει να αποθηκεύει την πιο πρόσφατη τιμή που έχει ένα object. Με αυτό τον τρόπο όταν υποβάλλεται ένα query, για το εν λόγω αντικείμενο, δεν χρειάζεται να προσπελάσουμε ολόκληρο το *Blockchain* για να βρούμε την τρέχουσα κατάστασή του. Αντίθετα, η πιο πρόσφατη ενημέρωση βρίσκεται στην βάση και το *Ledger* διατηρεί μόνο το ιστορικό των συναλλαγών. Με αυτό τον τρόπο αυξάνεται η επίδοση και ο χρόνος ανταπόκρισης του συστήματος. Σε επόμενη υπο-ενότητα θα εξετάσουμε ενδελεχώς την λειτουργία των βάσεων αυτών ή *World State* όπως τις ονομάζει το *Hyperledger Fabric*.

3.3.5 Υπηρεσία Σειριοποίησης

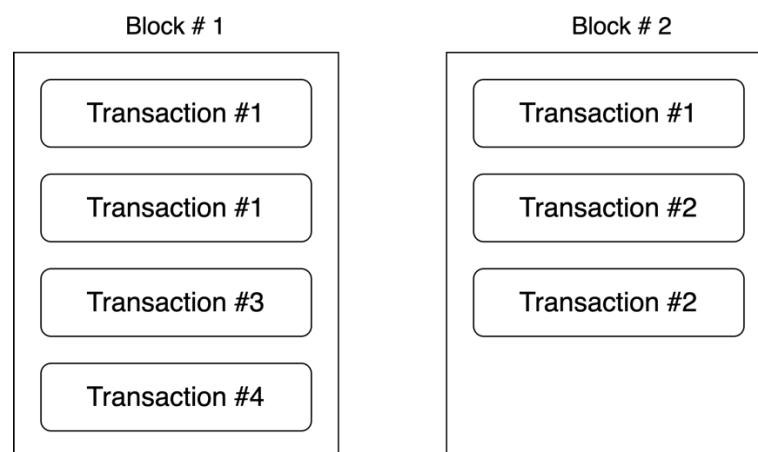
Αυτή η υποενότητα έχει σκοπό να μας εισάγει στην έννοια της υπηρεσίας σειριοποίησης (*Orderer*) και να μας δείξει με ποιο τρόπο οι συναλλαγές μπαίνουν σε *blocks* και πως το δίκτυο καταλήγει σε πλήρη συμφωνία μέσα από την διαδικασία *consensus*.

Σε διάφορα *Blockchain* συστήματα όπως το *Bitcoin* και το *Ethereum*, τα οποία δεν είναι *permissioned* όπως το *Hyperledger Fabric*, οι κόμβοι του δικτύου αναγκάζονται να συμμετάσχουν στην διαδικασία συναίνεσης (*consensus*). Δηλαδή κάθε κόμβος λαμβάνει μέρος στην ομαδοποίηση των *transaction* και στην δημιουργία του επόμενου *block*. Το

γεγονός αυτό, αναγκάζει τα συστήματα αυτά να χρησιμοποιούν πιθανοτικούς *consensus* αλγόριθμους όπως το *PwO* (*Proof Of Work*).

Το *Hyperledger Fabric* λειτουργεί με έναν διαφορετικό τρόπο. Συγκεκριμένα διαθέτει έναν ειδικό κόμβο ο οποίος ονομάζεται *Orderer*. Η κύρια υπευθυνότητα αυτού του κόμβου είναι να παραλαμβάνει συναλλαγές από τα *Peers* του δικτύου και να τις εσωκλείει σε *block* ανάλογα με την χρονοσφραγίδα τους. Σε μεταγενέστερο στάδιο αναμεταδίδει τα *block* αυτά σε όλα τα *Peers* του δικτύου ούτως ώστε να γίνουν κομμάτι του *Blockchain*. Ένα *Ordering Service* μπορεί να αποτελείται από πολλούς *Orderers* και συνεπώς όλοι αυτοί οι *Orderers* πρέπει να συμφωνούν μεταξύ τους για την τιμή του επόμενου *block*. Για αυτό το λόγο χρησιμοποιούνται ντετερμινιστικοί *consensus* αλγόριθμοι όπως είναι το *RAFT*. Το *RAFT* ακολουθεί το μοντέλο «*Leader and Follower*», όπου ένας κόμβος εκλέγεται ως αρχηγός και οι υπόλοιποι πρέπει να ακολουθούν τις αποφάσεις του.

Ο αριθμός των συναλλαγών που μπαίνουν σε ένα *block* εξαρτάται από τις ρυθμίσεις του δικτύου. Συγκεκριμένα μπορούμε να ελέγξουμε το μέγεθος του *block* βάση χρονικού διαστήματος ή του όγκου δεδομένων που έχει ήδη καταγραφεί σε αυτό. Στο ακόλουθο σχήμα βλέπουμε δύο πιθανά *block* που μπορούν να παραχθούν.

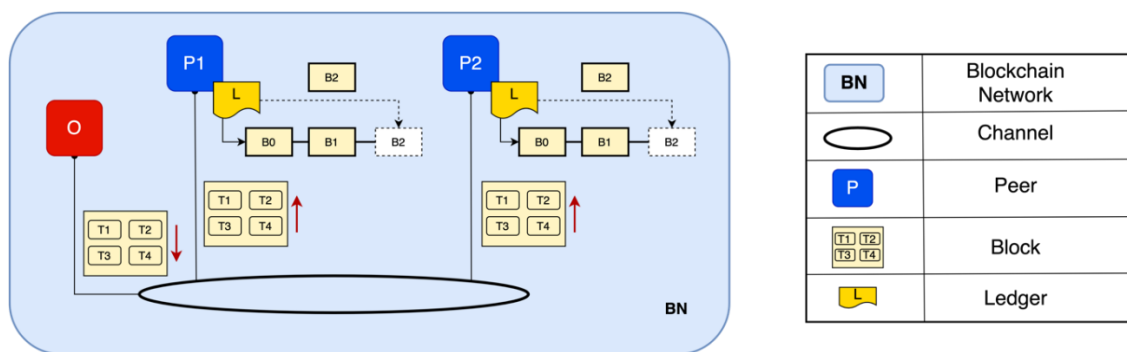


Max Block Size : 1 MB Block Time Interval : 2 sec

Σχήμα 3.9 Παράδειγμα γεμάτου και μισογεμάτου *Block*

Στην πιο πάνω περίπτωση έχουμε ρυθμίσει το δίκτυό μας έτσι ώστε να παράγονται *block* με μέγιστο μέγεθος 1 MB και τα *block* αυτά να αποδεσμεύονται το αργότερο κάθε 2 δευτερόλεπτα. Όποια από αυτές τις συνθήκες ικανοποιηθεί πρώτη σηματοδοτεί την αναμετάδοση του *block* στο δίκτυο. Για παράδειγμα, το *block* #1 έχει γεμίσει με συναλλαγές και το συνολικό του μέγεθος έχει φτάσει το 1 MB, ενώ το *block* #2 παρόλο που πέρασαν 2 δευτερόλεπτα δεν έχει γεμίσει ακόμα. Και τα δύο αυτά *block* είναι ορθά κατασκευασμένα και μπορούν προχωρήσουν στο στάδιο της διανομής.

Το ακόλουθο διάγραμμα σκιαγραφεί το τελευταίο βήμα όπου γίνεται η αναμετάδοση των *block* στα *Peers*.



Σχήμα 3.10 Παράδειγμα διανομής Block από τον Orderer στα Peers

3.3.6 Έξυπνα Συμβόλαια

Οι οργανισμοί ή οι επιχειρήσεις σε έναν εμπορικό κύκλο, πρέπει να ορίσουν ένα κοινό σύνολο από κανόνες και συμβόλαια προτού αρχίσουν να συναλλάσσονται. Οι συμφωνίες αυτές καθορίζουν το γενικό επιχειρηματικό μοντέλο που διέπει όλες τις αλληλεπιδράσεις μεταξύ των συναλλασσόμενων οργανισμών.

Σε γενικές γραμμές, ένα *smart contract* υλοποιεί σε μορφή προγράμματος τις συμφωνίες μεταξύ διάφορων εταίρων. Το πρόγραμμα αυτό εκτελείται στο *Blockchain* δίκτυο και διαχειρίζεται τα *objects* των οργανισμών. Για παράδειγμα, ένα *smart contract* μπορεί να διασφαλίζει ότι η παράδοση ενός πακέτου έχει γίνει σε συγκεκριμένο χρονικό διάστημα ή ότι η εξόφληση του προϊόντος έχει ήδη πραγματοποιηθεί. Με τον τρόπο αυτό πολλές

χειροκίνητες διαδικασίες όπου χρειαζόταν η επέμβαση φερέγγυου ατόμου, αυτοματοποιούνται, ενώ ταυτόχρονα διατηρείται η αξιοπιστία στο σύστημα.

Στο μοντέλο του *Hyperledger Fabric* τα *smart contracts* ονομάζονται και *chaincode*. Οι δύο αυτές ορολογίες χρησιμοποιούνται εναλλακτικά και έχουν το ίδιο νόημα.

Τα *smart contracts* είναι υπεύθυνα για την αλληλεπίδραση του χρήστη με το *Ledger*. Αναλυτικότερα, μπορούν να εκτελέσουν τις ακόλουθες λειτουργίες.

- *GET*: Χρησιμοποιείται για την εξαγωγή δεδομένων από το *Ledger* και το *World State* – υποβολή queries.
- *PUT*: Χρησιμοποιείται για την δημιουργία ή ενημέρωση κάποιου object στο *Ledger* και το *World State* – υποβολή transaction.
- *DELETE*: Χρησιμοποιείται για να αφαιρεθεί κάποιο object από το *Ledger*.

3.3.7 Καθολικό

Ιστορικά μιλώντας τα καθολικά (*Ledgers*) κάνουν την εμφάνισή τους σε αρχαίους πολιτισμούς όπως οι Σουμέριοι, οι οποίοι χρησιμοποιούσαν καθολικά για καταγραφή δεδομένων σχετικά με τις εμπορικές τους συναλλαγές.

Στην ψηφιακή εποχή στην οποία ζούμε, υπάρχει ανάγκη για καταγραφή πληροφοριών σχετικά με διάφορα object. Έτσι λοιπόν δημιουργήθηκαν τα ψηφιακά *Ledgers*. Σε ένα ψηφιακό *Ledger* αποθηκεύουμε τα γεγονότα και τις πληροφορίες σχετικά με ένα object έτσι ώστε να μπορούμε να προσδιορίσουμε την κατάστασή του. Η παρούσα κατάσταση ενός object μπορεί να αλλάξει αλλά το ιστορικό του παραμένει αμετάβλητο. Αυτή η έννοια που περιγράψαμε μπορεί να λάβει σάρκα και οστά μέσα από ένα *Blockchain*.

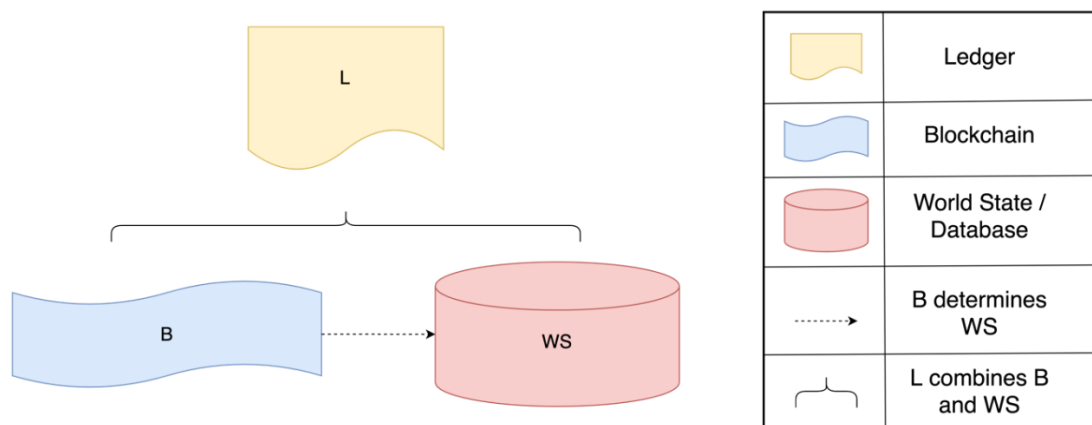
Στην περίπτωση του *Hyperledger Fabric*, το *Ledger* αποτελείται από δύο μέρη, το *World State* και το *Blockchain*.

Αρχικά έχουμε το *World State* το οποίο είναι μια βάση δεδομένων που διατηρεί τις πιο πρόσφατες τιμές διάφορων objects. Με αυτή την προσέγγιση μπορούμε να έχουμε άμεση πρόσβαση στα δεδομένα κάποιου object χωρίς να χρειάζεται να προσπελάσουμε ολόκληρο το *transaction log* για να υπολογίσουμε την κατάστασή του. Τα αντικείμενα

στο *Hyperledger Fabric* μπορούν να εκφραστούν ως ένα ζευγάρι key-value όπως γίνεται και στις παραδοσιακές βάσεις δεδομένων.

Το *Hyperledger Fabric* στην πιο πρόσφατη του έκδοση υποστηρίζει δύο βάσεις δεδομένων, την *CouchDB* και την *LevelDB*

Το δεύτερο σημαντικό κομμάτι του *Ledger* είναι το *Blockchain*. Το *Blockchain* όπως περιεγράφηκε σε προηγούμενη υποενότητα, μπορεί να θεωρηθεί ως ένα *transaction log*. Οι συναλλαγές ομαδοποιούνται σε *block* τα οποία προθέτονται στο *Blockchain*. Με αυτό τον τρόπο διασφαλίζεται η εγκυρότητα και η αξιοπιστία της κατάστασης στην οποία έχουν καταλήξει τα *objects* που έχουν αποθηκευτεί στο *Ledger*. Στο πιο κάτω σχήμα σκιαγραφείται η έννοια του *Ledger*.



Σχήμα 3.11 Αναπαράσταση του *Ledger* στο *Hyperledger Fabric*

Κεφάλαιο 4

Αρχιτεκτονική *TriaBase*

4.1 Η ανατομία της <i>TriaBase</i>.....	46
4.1.1 Τμήμα Χρήστη.....	47
4.1.1.1 Διεργασίες Τμήματος Χρήστη.....	47
4.1.1.2 Διαδικασία Ομοσπονδιακής Μάθησης.....	49
4.1.2 Τμήμα Αλυσίδας Συστοιχιών	50
4.1.2.1 Έξυπνο Συμβόλαιο RAT	51
4.1.2.3 Έξυπνο Συμβόλαιο MAT.....	54

Λαμβάνοντας υπόψη την έρευνα που έγινε στο πεδίο των αλυσίδων συστοιχιών (*Blockchain*) και στην αποθήκευση *IoT* δεδομένων, καταλήξαμε στην δημιουργία της *TriaBase*. Ο ορισμός αυτός προέρχεται από την λέξη «τρία», που παραπέμπει στο Web 3.0, και την λέξη «Base» που προφανώς αναφέρεται στις βάσεις δεδομένων. Η επερχόμενη τεχνολογική εξέλιξη μας φέρνει προ των πυλών για μια νέα εποχή η οποία θα έχει ως βάση την τρίτη γενεά Web. Κύριο χαρακτηριστικό του Web3.0, είναι ότι οι χρήστες θα μπορούν να έχουν τον έλεγχο των ψηφιακών τους στοιχείων αφαιρώντας τον κεντρικό έλεγχο που ασκούν μεγάλες εταιρείες όπως την Amazon και την Google [37]. Για να επιτύχουμε το ζητούμενο στηρίζομαστε στην τεχνολογία *Blockchain* και σε κατανεμημένα ψηφιακά *Ledger*. Αυτό προσπαθήσαμε να υλοποιήσουμε με την δημιουργία της *TriaBase*, μιας κατανεμημένης δομής δεδομένων η οποία χρησιμοποιεί την τεχνολογία *Blockchain* για την αποθήκευση *IoT* δεδομένων.

Δεδομένης κάποιας *Telco* τοπολογίας (τοπολογία πάροχων τηλεπικοινωνίας), η εργασία αυτή στοχεύει στην παροχή αποτελεσματικών τρόπων αποθήκευσης και αναπαραγωγής δεδομένων με την χρήση της αρχιτεκτονικής *TriaBase*. Κύριο μέρος αυτής της

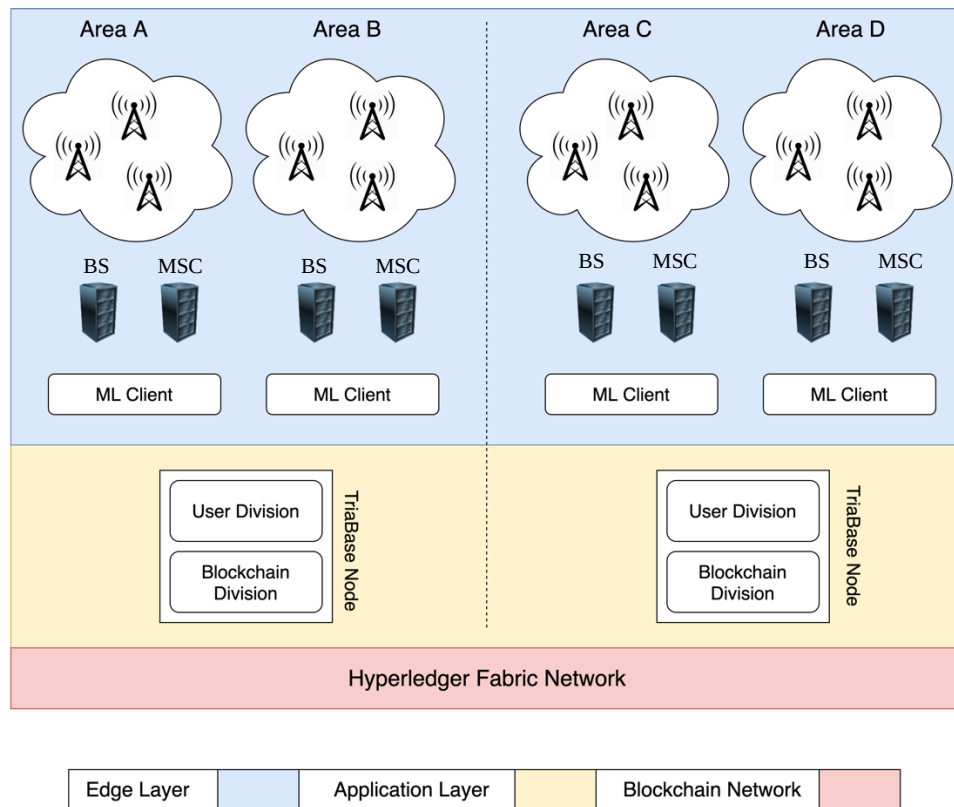
προσπάθειας είναι η κατανεμημένη διαχείριση μοντέλων μηχανικής μάθησης (*Machine learning*) έτσι ώστε να βελτιωθεί η εισαγωγή δεδομένων αλλά και η ικανότητα υποβολής επερωτήσεων σε αυτά.

Η τοπολογία του μοντέλου *TriaBase* παρουσιάζεται στο σχήμα 4.1 και αποτελείται από ένα τυπικό δίκτυο *Telco*. Ένα *Telco* δίκτυο χρησιμοποιείται ευρέως για την παροχή τηλεπικοινωνιακών υπηρεσιών ενώ ταυτόχρονα παρέχει την δυνατότητα στους διαχειριστές του να πραγματοποιούν αναλυτικές διεργασίες, σχετικές με τηλεπικοινωνιακά δεδομένα.

Αρχικά, σε πρώτο επίπεδο έχουμε το *Edge Layer*. Αυτό περιλαμβάνει το *Radio Network Layer*, όπου οι κεραίες είναι τοποθετημένες στον έξω κόσμο, ανταλλάζοντας δεδομένα σε πραγματικό χρόνο. Πιο αναλυτικά, σε ένα σύμπλεγμα από κεραίες 3G/4G/5G γεννιούνται πληροφορίες που αφορούν το ιστορικό κλήσεων, μηνύματα, συχνότητες εκπομπής και ούτω καθεξής. Τα δεδομένα αυτά είναι χρήσιμα για τους τηλεπικοινωνιακούς πάροχους, αφού προσφέρουν πολύτιμες πληροφορίες και στατιστικά. Η πληροφορία αυτή η οποία δημιουργείται ασταμάτητα και με μεγάλη ταχύτητα, μεταδίδεται στους *Base Station*, όπως φαίνεται και στο σχήμα 4.1. Σε αυτό το σημείο η επεξεργασία των δεδομένων δεν είναι εφικτή αφού μέχρι στιγμής τα δεδομένα είναι ένα μαύρο κουτί για εμάς.

Στο δεύτερο επίπεδο έχουμε τα *Mobile Switching Centers*, στα οποία είναι συνδεδεμένοι όλοι οι *Base Station*, και παρέχουν βασικές λειτουργίες όπως την δρομολόγηση τηλεφωνικών κλήσεων και μηνυμάτων.

Στη συνέχεια, δέσμες δεδομένων από τα *Mobile Switch Centers* καταλήγουν στους *ML Clients*, σε οριζόντια τμηματισμένα αρχεία για συγκεκριμένες χρονικές περιόδους. Ειδικότερα, κάθε οριζόντιο τμήμα του αρχείου αναπαριστά μια εγγραφή η οποία χαρακτηρίζεται από μια χρονοσφραγίδα. Σε αυτό το σημείο μπορούμε να εκπαιδεύσουμε διάφορα μοντέλα *Machine learning* χωρίς να χρειάζεται η μεταφορά των πραγματικών εγγραφών σε μια κεντρική υποδομή. Έπειτα τα μοντέλα αυτά, μαζεύονται και συσσωρεύονται (*aggregation*) στο τμήμα χρήστη (*User Division*) του επιπέδου εφαρμογής (*Application Layer*). Σε μεταγενέστερο στάδιο μπορούμε να εσωκλείσουμε τα μοντέλα αυτά σε δοσοληψίες (*transactions*) οι οποίες αποθηκεύονται στο τμήμα αλυσίδας συστοιχιών (*Blockchain Division*).

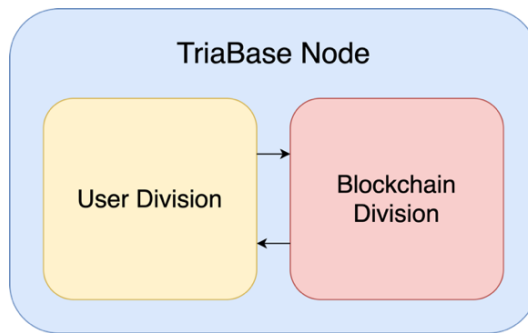


BS: *Base Station* MSC : *Mobile Switching Center*

Σχήμα.1 Αρχιτεκτονική TriaBase

4.1 Η ανατομία της *TriaBase*

Στη αρχιτεκτονική *TriaBase* και συγκεκριμένα στο *Application Layer* έχουμε δύο σημαντικά μέρη. Τα μέρη αυτά είναι το *User Division* και το *Blockchain Division*. Οι επόμενες δύο υποενότητες περιγράφουν αναλυτικότερα τις έννοιες αυτές. Στο σχήμα 4.2 βλέπουμε διαγραμματικά την δομή ενός κόμβου στην *TriaBase*.



Σχήμα 4.2 Δομή κόμβου TriaBase

4.1.1 Τμήμα Χρήστη

Ένας *Client* που τρέχει στο *User Division*, είναι υπεύθυνος για την παραλαβή και επεξεργασία δεδομένων από το *Edge*. Δηλαδή, σε αυτό το σημείο μπορούμε να μορφοποιήσουμε τα δεδομένα ανάλογα με τις ανάγκες μας. Ακόμη, σε αυτό το σημείο γίνεται ο έλεγχος των χρηστών του συστήματος. Ειδικότερα, κύριο χαρακτηριστικό της *TriaBase* είναι η παροχή δυνατότητας για άδειες (*Permissions*) που μπορεί να έχουν διάφοροι χρήστες. Οι *Clients* που τρέχουν στο *User Division* πρέπει να είναι εξουσιοδοτημένοι, αφού διαμέσω τους οι χρήστες είναι υπεύθυνοι να ενεργοποιήσουν μια σειρά από διεργασίες οι οποίες έχουν ως τελικό αποτέλεσμα την εισαγωγή δεδομένων ή την υποβολή επρωτήσεων στο *Blockchain Division*. Οι διαδικασίες αυτές περιγράφονται στην επόμενη υποενότητα. Επιπλέον, σημαντικό είναι να αναφέρουμε ότι, το *User Division* του *Application Layer* αναλαμβάνει τον ρόλο του «Server» στην τεχνική *Federated Learning*. Πιο αναλυτικά, είναι υπεύθυνο να συντονίζει τους *ML Clients* στα άκρα (*Edge Layer*) και να κάνει aggregate τα μοντέλα που εκπαιδεύουν, σε ένα καθολικό μοντέλο, που προωθείται στο *Blockchain Division*.

4.1.1.1 Διεργασίες Τμήματος Χρήστη

Στο *Application Layer* και ειδικότερα στο *User Division*, πραγματοποιούνται δύο διαφορετικές λειτουργίες. Συγκεκριμένα, εκεί γίνεται η διαχείριση όσον αφορά την εισαγωγή δεδομένων στο *Blockchain Division*, ως επίσης την υποβολή επρωτήσεων και την ανάκτηση δεδομένων. Οι λειτουργίες αυτές γίνονται πάντα υπό την αιγίδα κάποιου εξουσιοδοτημένου χρήστη, αφού όπως προαναφέρθηκε, η *TriaBase* είναι ένα *Permissioned Blockchain*. Πιο κάτω θα δούμε με μεγαλύτερη λεπτομέρεια τους

αλγόριθμους εισαγωγής (*Ingestion*) και επερωτήσεων (*Query*) οι οποίοι υλοποιούνται στο *User Division*.

Ingestion

1. Αρχικά το *User Division* αναλαμβάνει να εσωκλείσει τα δεδομένα που θέλουμε να προωθήσουμε στο *Blockchain Division*, σε ένα *Transaction Proposal*.
2. Το *Transaction Proposal* αποστέλνεται σε ένα ομότιμο κόμβο έγκρισης (*Endorsing Peer*), για να γίνει η ανάλογη προσομοίωση της δοσοληψίας. Εάν δεν υπάρχει οποιοδήποτε πρόβλημα στην προσομοίωση, το *Proposal* εγκρίνεται (γίνεται *endorse*) και επιστρέφεται η ανάλογη απάντηση.
3. Εν τω μεταξύ, ενεργοποιείται ένας *Event Listener*, ο οποίος εποπτεύει ένα κανάλι εκπομπής μηνυμάτων από τα *Peers*, έτσι ώστε να γνωρίζουμε πότε έγινε τελική επικύρωση (*Commit*) κάποιας δοσοληψίας.
4. Σε συνέχεια ένα *Endorsed Transaction* αποστέλλεται στην υπηρεσία σειριοποίησης (*Ordering Service*), όπου συμπεριλαμβάνεται σε ένα *block* με πολλές δοσοληψίες και διανέμεται σε όλα τα *Peer* του δικτύου.
5. Όταν η δοσοληψία φτάσει στον προορισμό της και γίνει *commit* στο τοπικό καθολικό (*Ledger*), ένα μήνυμα επιτυχίας εκπέμπεται στο κανάλι επικοινωνίας. Λαμβάνουμε το μήνυμα αυτό μέσα από τον *Event Listener* που είχαμε ενεργοποιήσει νωρίτερα.

Στον πίνακα 4.1 βλέπουμε συνοπτικά τα βήματα του αλγόριθμου εισαγωγής δεδομένων.

Αλγόριθμος : Εισαγωγή δεδομένων	
1:	Δημιουργία Tx Proposal με τα δεδομένα που θέλουμε να εισάγουμε στο Storage Layer
2:	Αποστολή Tx Proposal σε κάποιο <i>Endorsing Peer</i> για <i>Simulation/Execution</i>
3:	Αναμονή μέχρι να ληφθεί Tx Endorsement από το <i>Endorsing Peer</i>
4:	Αρχειοποίηση Event Listener
5:	Αποστολή του Endorsed Tx στο Ordering Service
6:	Δημιουργία Block από το Ordering Service
7:	Διανομή Block σε όλα τα <i>Peer</i> του δικτύου
8:	Commit δεδομένων στο local Ledger του <i>Peer</i>
9:	Αναμετάδοση Commit Event στο δίκτυο
10:	Λήψη Commit Event και τερματισμός Event Listener

Πίνακας 4.1 Εισαγωγή δεδομένων

Query

1. Αρχικά το *User Division* συνδέεται σε ένα τοπικό *Peer* το οποίο διατηρεί το *Ledger* στο οποίο θέλουμε να υποβάλουμε επερώτηση.
2. Ετοιμάζουμε αναλόγως την επερώτησή μας, δίνοντας τις σωστές παραμέτρους και μορφή. Η επερώτηση αυτή πρέπει να συμβαδίζει με την μορφή επερωτήσεων που δέχεται η *CouchDB* ή η *LevelDB* αναλόγως.
3. Στην συνέχεια ετοιμάζεται ένα αίτημα (*request*) το οποίο περιλαμβάνει το *query* και άλλες παραμέτρους.
4. Το *request* υποβάλλεται στο *Peer*.
5. Το *Peer* εκτελεί το *query* τοπικά έναντι του αντίστοιχου *Ledger*.
6. Το αποτέλεσμα επιστρέφεται στον χρήστη.

Στον πίνακα 4.2 βλέπουμε συνοπτικά τα βήματα του αλγορίθμου υποβολής επερωτήσεων.

Αλγόριθμος : Υποβολή επερωτήσεων	
1:	Σύνδεση σε τοπικό <i>Peer</i> που έχει αντίγραφο του <i>Ledger</i> που μας ενδιαφέρει
2:	Ετοιμασία <i>CouchDB/LevelDB query</i>
3:	Δημιουργία <i>request</i> προς το <i>Peer</i> . Το <i>request</i> έχει ως argument το <i>query</i>
4:	Υποβολή <i>request</i> στο <i>Peer</i>
5:	Εκτέλεση <i>query</i> έναντι του τοπικού <i>Ledger</i> και δημιουργία απάντησης
6:	Επιστροφή απάντησης στον χρήστη

Πίνακας 4.2 Υποβολή επερωτήσεων

4.1.1.2 Διαδικασία Ομοσπονδιακής Μάθησης

Στα πλαίσια αυτής της διπλωματικής εργασίας, προτείνεται η χρήση της τεχνικής *Federated Learning*, όπως αυτή περιγράφεται στο κεφάλαιο 2, για να ενισχύσουμε την *Blockchain* δομή μας.

Στην αρχιτεκτονική *TriaBase*, η εκπαίδευση των μοντέλων *Machine learning* λαμβάνει χώρα στους *ML Clients*, στο *Edge*, και το *User Division* αναλαμβάνει να δημιουργήσει κάποια συγκεντρωτικά μοντέλα. Τα συγκεντρωτικά μοντέλα αποθηκεύονται στο

Blockchain Division, το οποίο με την σειρά του διανέμει αυτά τα μοντέλα σε όλο το δίκτυο *Blockchain*. Στον πίνακα 4.3 φαίνονται αναλυτικότερα τα βήματα τα οποία ακολουθούνται για την δημιουργία κάποιου μοντέλου στην *TriaBase*.

Αλγόριθμος : <i>Federated Learning</i> Process	
1:	Υποθέτουμε ότι οι χρήστες $U = U_1, U_2, U_3 \dots U_N$ διατηρούν κόμβους <i>Federated Learning – ML Clients</i>
2:	for each epoch e do
3:	for each iteration round r of user U_i do
4:	U_i εκτελεί τοπικούς υπολογισμούς για την δημιουργία ενός μοντέλου με τα τοπικά δεδομένα d_i
5:	U_i ενημερώνει το server state μοντέλο και τις τοπικές παραμέτρους
6:	U_i αποστέλλει το νέο μοντέλο στο κοντινό «Server» (<i>User Division</i>)
7:	end for
8:	end for

Πίνακας 4.3 Διαδικασία Federated Learning

Μετά το βήμα 6, που περιγράφεται στον πιο πάνω αλγόριθμο, ακολουθείται ο αλγόριθμος εισαγωγής δεδομένων όπως αυτός περιγράφεται στην υποεόντητα «*User Division Operations*»

4.1.2 Τμήμα Αλυσίδας Συστοιχιών

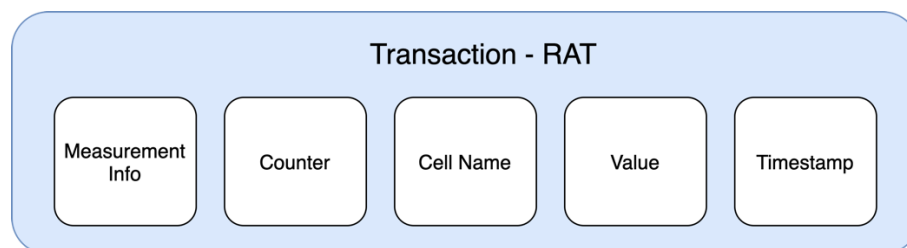
Το *Blockchain Division* αποτελεί ουσιαστικά τον τομέα αποθήκευσης των δεδομένων. Στην περίπτωση μας το *Blockchain Division* υλοποιείται από το *framework Hyperledger Fabric*. Το δίκτυό μας αποτελείται κυρίως από *Endorsing Peers*, από κανονικά *Peers* και το *Ordering Service*. Το *Ordering Service* αναλαμβάνει την δημιουργία *block* που περιέχει τις δοσοληψίες, που έχουν σταλεί από το *User Division*, και σηματοδοτεί την έναρξη του *consensus* πρωτόκολλου στο δίκτυο. Ακόμη υποθέτουμε ότι στο δίκτυό μας λαμβάνουν μέρος πολλοί οργανισμοί, οι οποίοι διατηρούν ο καθένας τουλάχιστον ένα *Endorsing Peer*, και πολλαπλά *Peers*. Επίσης, υποθέτουμε ότι σε κάθε *Endorsing Peer* τρέχει κάποιο είδος έξυπνου συμβολαίου (*smart contract*), το οποίο αναλαμβάνει την προσομοίωση και την εκτέλεση της συναλλαγής. Στις επόμενες δύο υποεονότητες θα δούμε σε λεπτομέρεια τον τρόπο λειτουργίας των *smart contracts* για διαφορετικές

προσεγγίσεις αποθήκευσης δεδομένων. Σημαντικό είναι να τονίσουμε ότι στην αρχιτεκτονική *TriaBase*, η προσέγγιση η οποία προτείνεται είναι η MAT, η οποία πραγματεύεται την χρήση μοντέλων *Machine learning* για την διαχείριση δεδομένων.

Συνοψίζοντας, κύριες λειτουργίες του *Blockchain Division* είναι η διατήρηση του *Ledger*, το *endorsement* νέων *transactions* που υποβάλλονται από το *User Division* και η μερική συμμετοχή στην διαδικασία *consensus* στο δίκτυο.

4.1.2.1 Έξυπνο Συμβόλαιο RAT

Η προσέγγιση RAT αναφέρεται στην αγγλική ορολογία *Record At a Time*, όπου κάθε *Blockchain transaction* αντιπροσωπεύει μια εγγραφή τηλεπικοινωνιακών δεδομένων. Αναφερόμαστε στην προσέγγιση RAT θέλοντας να δείξουμε πως θα λειτουργούσε μια συμβατική *Blockchain* αρχιτεκτονική, με σκοπό να υπάρξει κάποια σύγκριση με την *TriaBase* σε επόμενο κεφάλαιο. Παράδειγμα μιας εγγραφής που αφορά την προσέγγιση RAT φαίνεται πιο κάτω.



Σχήμα 4.3 Παράδειγμα Transaction για την προσέγγιση RAT

Τα στοιχεία που εμπεριέχονται στην πιο πάνω εγγραφή είναι το *Measurement Info*, το *Counter*, το *Cell Name*, το *Value* και το *Timestamp* αντίστοιχα.

Στην επόμενη σελίδα βλέπουμε τον αντίστοιχο κώδικα που χρησιμοποιήθηκε για την υλοποίηση των βασικών λειτουργιών στην προσέγγιση RAT.

Σημείωση : ο κώδικας που περιγράφεται πιο κάτω είναι γραμμένος σε *golang* και αποτελεί κομμάτι του κώδικα ο οποίος βρίσκεται στο Παράρτημα «B».

Telco struct

- **Γραμμές 1 – 7** : Αυτό το κομμάτι βρίσκεται στην κεφαλίδα του *Smart contract* και περιγράφει την μορφή μιας εγγραφής που θέλουμε να εισάγουμε στο *Ledger*. Η εγγραφή αυτή περιέχει γνωρίσματα (*attributes*), όπως αυτά έχουν περιγραφεί πιο πάνω και οι τύποι δεδομένων τους είναι *string*, *string*, *string*, *float32* και *int64* αντίστοιχα.

Συνάρτηση write

- **Γραμμές 9 – 10** : Σε αυτές τις γραμμές βλέπουμε το *function signature* για την συνάρτηση *write*. Λαμβάνει ως είσοδο το *stub* για να μπορεί να έχει πρόσβαση στο *Ledger*, ως επίσης ένα πίνακα τύπου *string*, ο οποίος περιέχει όλες τις παραμέτρους που έδωσε ο χρήστης. Η συνάρτηση *write* επιστρέφει μια απάντηση τύπου «*Peer.response*» η οποία ουσιαστικά είναι μια ακολουθία από *bytes*.
- **Γραμμές 12 – 34** : Εδώ γίνεται επεξεργασία των τιμών εισόδου που βρίσκονται μέσα στο *string array*. Σε περίπτωση σφάλματος ή δοθέντων λανθασμένων παραμέτρων, η συνάρτηση τερματίζεται και επιστρέφεται το ανάλογο μήνυμα.
- **Γραμμές 36 – 43** : Δημιουργία κλειδιού/ID και εγγραφής, σε *struct*, με όλα τα σχετικά δεδομένα.
- **Γραμμές 45 – 48** : Μετατροπή *struct* σε *JSON bytes*.
- **Γραμμές 50 – 55** : Εγγραφή ζευγαριού ID/*JSON bytes* στο *Ledger* και επιστροφή κατάλληλου μηνύματος.

Συνάρτηση queryRecords

- **Γραμμές 58 – 59** : Σε αυτές τις γραμμές βλέπουμε το *function signature* για την συνάρτηση *queryRecords*. Λαμβάνει ως είσοδο το *stub* για να μπορεί να έχει πρόσβαση στο *Ledger*, ως επίσης ένα πίνακα τύπου *string*, ο οποίος περιέχει όλες τις παραμέτρους που έδωσε ο χρήστης. Η συνάρτηση *queryRecords* επιστρέφει μια απάντηση τύπου «*Peer.response*» η οποία ουσιαστικά είναι μια ακολουθία από *bytes*.
- **Γραμμές 61 – 65** : Γίνεται έλεγχος για τα δεδομένα τα οποία δόθηκαν ως είσοδος από τον χρήστη. Αφού επικυρωθούν, εξάγουμε την πρώτη τιμή η οποία είναι το ID της εγγραφής την οποία θέλουμε να αναζητήσουμε στο *Ledger*.

- **Γραμμές 67 – 74 :** Αναζήτηση συγκεκριμένου ID στο *Ledger* και επιστροφή της τιμής του σε *bytes*. Επίσης, γίνεται έλεγχος σε περίπτωση που επιστραφεί κάποιο σφάλμα ή σε περίπτωση που η εγγραφή δεν υπάρχει. Σε κάθε περίπτωση επιστρέφεται η ανάλογη απάντηση. Τέλος, αν όλα γίνουν ορθά, επιστρέφεται στο χρήστη η σχετική εγγραφή σε *bytes*.

```

1  type TelcoEntry struct {
2      MeasInfo  string  'json:"meas_info"'
3      Counter   string  'json:"counter"'
4      CellName  string  'json:"cell_name"'
5      Value     float32  'json:"value"'
6      Timestamp int64   'json:"timestamp"'
7  }
8
9  func (s *SmartContract) write(stub shim.ChaincodeStubInterface, args []string) peer.Response {
10
11      if len(args) != 5 {
12          return shim.Error("Invalid number of arguments.")
13      }
14
15      meas_info := strings.ToLower(args[0])
16
17      counter := strings.ToLower(args[1])
18
19      cell_name := strings.ToLower(args[2])
20      if (len(cell_name) != 32){
21          return shim.Error("Cell name must have 32 characters length")
22      }
23
24      v, err := strconv.ParseFloat(args[3], 32)
25      if err != nil {
26          return shim.Error("4th argument must be a numeric string")
27      }
28      value := float32(v)
29
30      timestamp, err := strconv.ParseInt(args[4], 10, 64)
31      if err != nil {
32          return shim.Error("5th argument must be a numeric string")
33      }
34
35      record_id := meas_info + "_" + counter + "@"M-^\_âM-^@M-^] + timestamp
36      record := TelcoEntry{
37          MeasInfo: meas_info,
38          Counter: counter,
39          CellName: cell_name,
40          Value : value,
41          Timestamp : timestamp,
42      }
43
44      recordAsBytes, err := json.Marshal(record)
45      if err != nil {
46          return shim.Error(err.Error())
47      }
48
49      err = stub.PutState(record_id, recordAsBytes)
50      if err != nil {
51          return shim.Error(err.Error())
52      }
53
54      return shim.Success(nil)
55  }
56
57  func (s *SmartContract) queryRecords(stub shim.ChaincodeStubInterface, args []string) peer.Response {
58
59      if len(args) < 1 {
60          return shim.Error("Incorrect number of arguments. Expecting only
61      1")

```

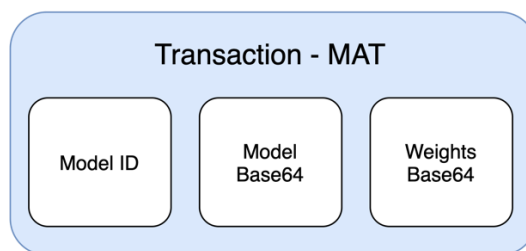
```

61     }
62     id := args[0]
63     result, err := stub.GetState(id)
64     if err != nil {
65         return shim.Error(err.Error())
66     } else if result == nil {
67         jsonResp = "{\"Error\":\"Entry does not exist: \" + id + \"\"}"
68         return shim.Error(jsonResp)
69     }
70     return shim.Success(result)
71 }
72 }
73 }

```

4.1.2.3 Έξυπνο Συμβόλαιο MAT

Η προσέγγιση MAT αναφέρεται στην αγγλική ορολογία *Model At a Time*, όπου κάθε *Blockchain transaction* αντιπροσωπεύει ένα μοντέλο *Machine learning*. Παράδειγμα μιας τέτοιας εγγραφής φαίνεται πιο κάτω.



Σχήμα 4.4 Παράδειγμα Transaction για την προσέγγιση MT

Τα στοιχεία που εμπεριέχονται στο πιο πάνω *transaction* είναι το ID, Model_Base64 και Weights_Base64 αντίστοιχα. Η μορφή *Base64* χρησιμοποιείται για τον λόγο ότι τα μοντέλα *Machine learning* είναι σε *binary* μορφή και χρειάζεται να μετατραπούν σε εκτυπώσιμους χαρακτήρες έτσι ώστε να αποθηκευτούν στο *Ledger*.

Στην επόμενη σελίδα βλέπουμε τον αντίστοιχο κώδικα που χρησιμοποιήθηκε για την υλοποίηση των βασικών λειτουργιών στην προσέγγιση MAT.

Σημείωση : ο κώδικας που περιγράφεται πιο κάτω είναι γραμμένος σε *NodeJS* και αποτελεί κομμάτι του κώδικα ο οποίος βρίσκεται στο Παράρτημα «B».

Συνάρτηση writeModel

- **Γραμμή 7 :** Εδώ βλέπουμε το *function signature* για την συνάρτηση writeModel. Λαμβάνει ως είσοδο το ctx για να μπορεί να έχει πρόσβαση στο Ledger, ως επίσης μια σειρά από παραμέτρους που έδωσε ο χρήστης. Η συνάρτηση writeModel επιστρέφει σε Bytes, απάντηση σχετικά με την επιτυχία εγγραφής της στο Ledger.
- **Γραμμές 9 – 13 :** Εδώ δημιουργείται ένα JavaScript object το οποίο αντιπροσωπεύει την εγγραφή ενός μοντέλου.
- **Γραμμή 15 :** Εγγραφή ζευγαριού ID/JSON bytes στο Ledger και επιστροφή κατάλληλου μηνύματος.

Συνάρτηση queryModel

- **Γραμμή 19 :** Εδώ βλέπουμε το *function signature* για την συνάρτηση queryModel. Λαμβάνει ως είσοδο το ctx για να μπορεί να έχει πρόσβαση στο Ledger, ως επίσης ό,τι έδωσε ο χρήστης. Η συνάρτηση queryModel επιστρέφει σε Bytes, απάντηση σχετικά με την επιτυχία εγγραφής της στο Ledger.
- **Γραμμές 21 – 24 :** Γίνεται έλεγχος για τα δεδομένα τα οποία δόθηκαν ως είσοδος από τον χρήστη και εξάγεται από το Ledger το μοντέλο με το ανάλογο ID. Σε περίπτωση σφάλματος επιστρέφεται το ανάλογο μήνυμα.
- **Γραμμές 26 – 61 :** Το μοντέλο που ανακτήθηκε μετατρέπεται σε JSON και ακολούθως τα JSON bytes επιστρέφονται ως αποτέλεσμα.

```
1 class contract_mlModel extends Contract {
2
3     async Init(ctx) {
4
5     }
6
7     async writeModel(ctx, id, model_json, model_weights) {
8
9         const Telco_Model = {
10             ID: id,
11             Model_Json: model_json,
12             Model_Weights: model_weights,
13         };
14
15         return ctx.stub.putState(id, Buffer.from(JSON.stringify(Telco_Model)));
16
17     }
18
19     async queryModel(ctx, id) {
20
21         const Telco_Model = await ctx.stub.getState(id);
22         if (!Telco_Model || Telco_Model.length === 0) {
23             throw new Error('The Model ${id} does not exist');
24         }
25
26         var record = JSON.parse(Telco_Model)
27
28         var model_buffer = new Buffer(record.Model_Json, 'base64');
29         var weights_buffer = new Buffer(record.Model_Weights, 'base64');
30
31         return model_buffer.toString();
32     }
33 }
```

Κεφάλαιο 5

Πειραματική Μεθοδολογία

5.1 Δίκτυο Δομικών.....	56
5.2 Δεδομένα	57
5.3 Πειραματική διαδικασία	59

Σε αυτό το κεφάλαιο περιγράφεται η σειρά πειραμάτων και οι προσομοιώσεις που διενεργήθηκαν. Στα πλαίσια αυτής της διπλωματικής εργασίας, σκοπός των πειραμάτων είναι να ελέγξουμε την απόδοση της προτεινόμενης αρχιτεκτονικής *TriaBase* όσον αφορά το τμήμα αλυσίδας συστοιχών (*Blockchain Division*). Πιο αναλυτικά, θα αντιπαραβάλλουμε την προσέγγιση MAT, έναντι RAT, όπως αυτές περιγράφονται στο κεφάλαιο 4. Ακόμη, σε αυτό το κεφάλαιο θα περιγράψουμε την πειραματική διάταξη, ως επίσης τα δεδομένα που χρησιμοποιήθηκαν.

5.1 Δίκτυο Δομικών

Σε αυτή την υποενότητα θα δούμε το πειραματικό δίκτυο το οποίο χρησιμοποιήθηκε για να γίνουν τα πειράματα. Πιο αναλυτικά, στο *Blockchain Network* έχουμε στήσει ένα *Hyperledger Fabric Network* το οποίο αποτελείται από δύο οργανισμούς, όπου κάθε οργανισμός διατηρεί δύο ομότιμους κόμβους (*Peers*) – ένα εκ των οποίων είναι κόμβος έγκρισης (*Endorsing Peer*) – και μια υπηρεσία σειριοποίησης (*Ordering Service*) με έναν *Orderer*. Το δίκτυο τρέχει τοπικά σε μια ιδεατή μηχανή στο Data Center του Data Management System Laboratory στο Πανεπιστήμιο Κύπρου. Επίσης, σημαντικό είναι να αναφέρουμε ότι οι κόμβοι του *Hyperledger Fabric* εσωκλείονται σε *Docker Containers*

έτσι ώστε να είναι πιο εύκολη η διαχείρισή τους καθώς επίσης και η επικοινωνία τους σε ένα ιδιωτικό ιδεατό *Peer-to-Peer* δίκτυο. Στην πιο κάτω αναφορά του Docker, παρουσιάζονται συνοπτικά όλοι οι κόμβοι του δικτύου *Hyperledger Fabric* που στήθηκε.

CONTAINER ID	IMAGE	PORTS	NAMES
3dc7eb773566	hyperledger/fabric-orderer:latest	0.0.0.0:7050->7050/tcp	orderer.example.com
c2eef6429894	hyperledger/fabric-peer:latest	7051/tcp, 0.0.0.0:8051->8051/tcp	peer1.org1.example.com
2ca2c906608e	hyperledger/fabric-peer:latest	7051/tcp, 0.0.0.0:10051->10051/tcp	peer1.org2.example.com
6cc751dd6855	hyperledger/fabric-peer:latest	0.0.0.0:7051->7051/tcp	peer0.org1.example.com
f517c30e3e72	hyperledger/fabric-peer:latest	7051/tcp, 0.0.0.0:9051->9051/tcp	peer0.org2.example.com

Σχήμα 5.1 Docker Report after executing the command " docker ps "

Για την υλοποίηση αυτού του δικτύου έχουν συγγραφεί shell scripts τα οποία επισυνάπτονται στο Παράρτημα «Β». Επιπλέον, στο Παράρτημα «Α» βρίσκεται αναλυτικός οδηγός (*tutorial*), τον οποίο μπορεί κανείς να χρησιμοποιήσει για να στήσει το δικό του *Hyperledger Fabric Network*.

Οι πειραματικές δοκιμές έγιναν στο DMSL VCenter IaaS Datacenter, το οποίο περιλαμβάνει 5 IBM System x3550 M3 και HP Proliant DL 360 G7 servers, οι οποίοι έχουν δυνατότητες για single (8 cores), ή double (16 cores) socket Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, αντίστοιχα. Οι server αυτοί έχουν συγκεντρωτικά 300 GB κύριας μνήμης και 16TB RAID-5 δευτερεύουσας μνήμης. Επιπλέον, το datacenter διαχειρίζεται μέσω VMWare vCenter Server 5.1 που συνδέεται με τους αντίστοιχους κεντρικούς υπολογιστές VMWare ESXi 5.0.0.

Οι υπολογιστικοί πόροι που χρησιμοποιήθηκαν στην πειραματική μας διάταξη είναι 120GB δευτερεύουσας μνήμης, 8 GB RAM και 8 ιδεατοί επεξεργαστές. Επίσης, το λειτουργικό σύστημα το οποίο χρησιμοποιήθηκε για να φιλοξενήσει την πειραματική διάταξη είναι το Ubuntu 18.04.

5.2 Δεδομένα

Τα δεδομένα που χρησιμοποιήθηκαν για την διεξαγωγή πειραμάτων, όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, σχετίζονται με τον τομέα των τηλεπικοινωνιών.

Αναλυτικότερα, πρόκειται για δεδομένα που συλλέχθηκαν από 1192 πύργους κινητής τηλεφωνίας που βρίσκονται διασκορπισμένοι σε μια γεωγραφική περιοχή της τάξης των 5,896 Km² . Το σύνολο δεδομένων (dataset) εμπεριέχει περίπου 100M ανώνυμες τηλεπικοινωνιακές εγγραφές για διάφορους δείκτες (Key Performance Indicators - KPIs) ή αλλιώς μετρητές (counters). Ειδικότερα, κάθε εγγραφή έχει τέσσερα κύρια χαρακτηριστικά τα οποία φαίνονται στον πιο κάτω πίνακα.

Telco Record Format		
Description	Type	Size (in Bytes)
Timestamp	int64	8
Measurement Information	string	8
Counter	string	8
Value	float32	4
Total Record Size		28

Πίνακας 5.1 Telco Record Format

Επιπρόσθετα στον πιο πάνω πίνακα βλέπουμε τον τύπο κάθε στοιχείου, το μέγεθός του και τον συνολικό όγκο δεδομένων που έχει η κάθε εγγραφή.

Τα πιο πάνω δεδομένα θα χρησιμοποιηθούν για τις δοκιμές που αφορούν την προσέγγιση RAT. Σχετικά με την προσέγγιση MAT, τα πιο πάνω δεδομένα χρησιμοποιήθηκαν για την εκπαίδευση μοντέλων *Machine Learning*, όπως αυτά περιεγράφηκαν και υλοποιήθηκαν στην σχετική δουλειά «Decaying Telco Big Data with Data Postdiction». Τα μοντέλα αυτά είναι τύπου *LSTM* και αποτελούνται από δύο κρυφές στρώσεις νευρώνων 16x16, ενώ το *Decay Factor* τους είναι 50%. Για την εκπαίδευση ενός τέτοιου μοντέλου χρησιμοποιούμε 1000 Telco εγγραφές και το αποτέλεσμα είναι να παραχθεί ένα μοντέλο όπως φαίνεται στον πιο κάτω πίνακα. Ο κώδικας που χρησιμοποιήθηκε για την υλοποίηση των *LSTM* νευρωνικών δικτύων βρίσκεται στο Παράρτημα «B».

<i>Machine learning Model Format</i>		
Description	Type	Size (in Bytes)
Model Structure JSON	string	70000
Model Weights	string	5000
Total Record Size		75000

Πίνακας 5.2 Machine learning Model Format

Όπως παρατηρούμε στον πιο πάνω πίνακα τα χαρακτηριστικά του μοντέλου έχουν μετατραπεί σε συμβολοσειρά. Αυτό έγινε για τον λόγο ότι στο εργαλείο *Blockchain* που χρησιμοποιήσαμε, *Hyperledger Fabric*, η αποθήκευση αρχείων σε δυαδική μορφή δεν είναι εφικτή. Συνεπώς χρειάστηκε να μετατρέψουμε τα μοντέλα σε εκτυπώσιμη μορφή χρησιμοποιώντας την με την χρήση κωδικοποίησης *Base64*.

5.3 Πειραματική διαδικασία

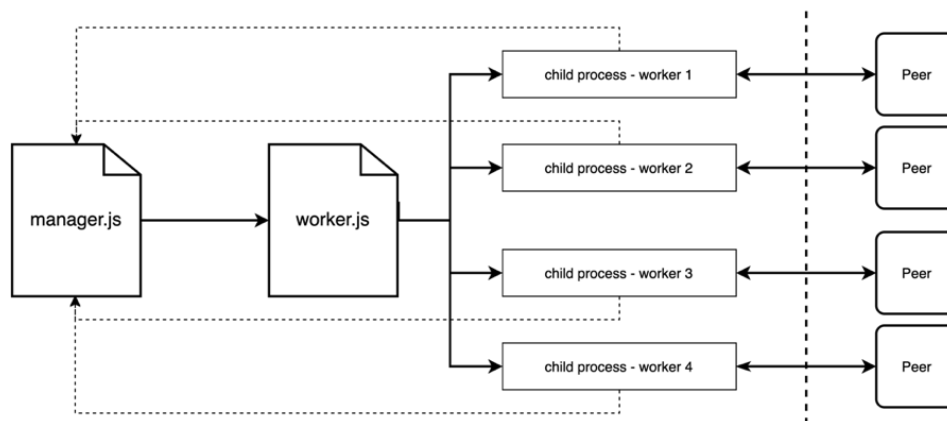
Για την διεξαγωγή των πειραμάτων υλοποιήθηκαν τα *smart contract* για τις δύο προσεγγίσεις που θα συγκρίνουμε, RAT και MAT.

Ο κώδικας των *smart contract* είναι γραμμένος σε *golang* για την προσέγγιση RAT, ενώ για την περίπτωση του MAT σε *NodeJS*. Η επιλογή της *golang* έγινε λαμβάνοντας υπόψη ότι είναι η επικρατέστερη γλώσσα προγραμματισμού στην οποία έχει γραφτεί και τρέχει ο κορμός του *Hyperledger Fabric*. Πέραν τούτου, η *golang* είναι ευρέως διαδεδομένη μεταξύ της κοινότητας προγραμματιστών *chaincode*, γεγονός που συνεπάγεται στην καλύτερη και άμεση υποστήριξη σε διάφορα προγραμματιστικά φόρουμ όπως το «Stack Overflow». Για την προσέγγιση MAT, το έξυπνο συμβόλαιο (*smart contract*) έχει υλοποιηθεί με την χρήση *NodeJS* αφού οι πλείστες *Machine learning* βιβλιοθήκες είναι γραμμένες σε *Python* ή *NodeJS*. Έτσι λοιπόν, δεδομένης της δυνατότητας εγγραφής κώδικα σε *NodeJS*, με τη χρήση του *Hyperledger Fabric Node SDK*, προτιμήσαμε να ακολουθήσουμε αυτό τον δρόμο. Εκτός από αυτό, σε μελλοντικό στάδιο αν θελήσουμε

να επεκτείνουμε την διαδικασία επεξεργασίας των μοντέλων σε επίπεδο *smart contract* θα είναι πιο εύκολο.

Ακόμη, για την αποτίμηση του *Blockchain Division* χρειάστηκε να δημιουργήσουμε δικό μας εργαλείο το οποίο προσομοιώνει την λειτουργία κάποιου *Client* όπως αυτός περιγράφεται στο Κεφάλαιο 4. Πιο αναλυτικά, κάθε *Client* μπορεί να ενωθεί, με την χρήση κάποιων διαπιστευτηρίων (*credentials*), σε ένα τοπικό *Peer* του *Hyperledger Fabric* και να υποβάλει αιτήματα για εγγραφή δεδομένων ή για την εκτέλεση επερωτήσεων έναντι του τοπικού *Ledger*.

Ειδικότερα, στην περίπτωση μας όπου έχουμε δύο οργανισμούς και τέσσερα *Peer*, θα έχουμε τέσσερεις *Clients* οι οποίοι θα υποβάλλουν στο σύστημα *transactions*. Για την αρχικοποίηση και έλεγχο των *Client*, αυτών έχει δημιουργηθεί, με την χρήση της γλώσσας προγραμματισμού NodeJS, ένα πρόγραμμα με το όνομα *manager.js*. Κύρια λειτουργία του προγράμματος αυτού είναι η γένεση και επίβλεψη των *Clients* - *child processes* – λαμβάνοντας στατιστικά και αναφορές από αυτούς. Στο σχήμα 5.2 βλέπουμε καλύτερα τον τρόπο με τον οποία το *manager.js* δημιουργεί και ελέγχει τους *Clients*(workers). Στο Παράρτημα «B» παρουσιάζεται ο κώδικας για το *manager.js* ως επίσης για τον εργάτη (*worker*).



Σχήμα 5.2 Benchmarking Tool

Επιλέξαμε να μετρήσουμε την απόδοση του *Blockchain Division* όσον αφορά τις μετρικές *Throughput* και *Latency*. Με τον όρο *Throughput* εννοούμε πόσες δοσοληψίες

(*transactions*) ανά δευτερόλεπτο μπορεί να χειριστεί το *Blockchain Division*. Αντίστοιχα, με τον όρο *Latency* εννοούμε τον χρόνο που χρειάζεται ένα *transaction* από την στιγμή που θα εκδοθεί στον *Client*, μέχρι να γίνει η τελική επικύρωση (*commit*) στο *Ledger* του κάθε *Peer*. Το σύστημα θα δοκιμαστεί ως προς την εισαγωγή δεδομένων αλλά και υποβολή επερωτήσεων, για διάφορους ρυθμούς αποστολής *transactions*. Τέλος, θα συγκρίνουμε τις δύο προσεγγίσεις ως προς τον συνολικό όγκο εγγραφών, που κατάφεραν να εισάγουν στο σύστημα ανάλογα με την χρήση διαφορετικού *World State*. Πιο κάτω βλέπουμε αναλυτικότερα τον τρόπο με τον οποίο μετρούμε το *Throughput*, *Latency* για τα σενάρια εισαγωγής δεδομένων και υποβολής επερωτήσεων.

Ingestion:

Ένα *child process* όπως φαίνεται στο σχήμα 5.2, δημιουργεί ένα σωρό από *transactions* βάση της παραμέτρου *Send Rate* (ρυθμός αποστολής). Για κάθε *transaction* που δημιουργείται υπάρχει ένα χρονόμετρο το οποίο μετρά τον χρόνο που χρειάστηκε το συγκεκριμένο *transaction* να γίνει *commit* στο *Ledger*. Αφού μετρήσουμε όλους αυτούς του χρόνους υπολογίζουμε το *Latency* βρίσκοντας τον μέσο όρο τους. Επιπλέον, το *Throughput* υπολογίζεται με το πηλίκο του μέγιστου χρόνου εκτέλεσης n *transactions* δια το *Send Rate*. Πιο κάτω βλέπουμε καλύτερα την διαδικασία που μόλις περιγράψαμε με μαθηματικό φορμαλισμό.

$$Latency = \frac{\sum_{i=1}^n f(i)}{n}$$

όπου n ο αριθμός του *Send Rate* και $f(i)$ το *latency* του i – οστού *transaction*

$$Throughput = \frac{Transaction\ Send\ Rate}{Total\ time\ to\ execute\ n\ transactions}$$

όπου n ο αριθμός *transaction* του *Send Rate*

Query:

Για τον υπολογισμό του *Latency* στην διαδικασία του *Query* μετρούμε τον χρόνο που χρειάζεται από την στιγμή που υποβάλλουμε ένα αίτημα στο *Blockchain Division* μέχρι την στιγμή που θα μας επιστραφεί μια συγκεκριμένη *Telco* εγγραφή. Στην περίπτωση *RAT* ο χρόνος αυτός εξαρτάται αποκλειστικά από το *Hyperledger Fabric*, ενώ στην περίπτωση *MAT*, το *Latency* είναι συνδυασμός του χρόνου ανάκτησης του μοντέλου από

το *Blockchain* σε συνδυασμό με τον χρόνο που χρειάζεται ένα μοντέλο για να κάνει μια πρόβλεψη. Όσον αφορά το *Throughput* της προσέγγισης RAT, αυτό υπολογίζεται με τον ίδιο τρόπο που το υπολογίζουμε στο *Ingestion*. Στην περίπτωση της προσέγγισης MAT το *Throughput* είναι το καρτεσιανό γινόμενο των μοντέλων που μπορούμε να ανακτήσουμε από το *Hyperledger Fabric* ανά δευτερόλεπτο, επί τον αριθμό των προβλέψεων που μπορεί να κάνει ένα μοντέλο, σε ένα δευτερόλεπτο.

$$Latency_{RAT} = \frac{\sum_{i=1}^n f(i)}{n}, \text{ όπου } f(i) = \text{model retrieval time}$$

$$Latency_{MAT} = \frac{\sum_{i=1}^n f(i)}{n}, \text{ όπου } f(i) = \text{model retrieval time} + \text{prediction time}$$

$$Throughput_{MAT} = \frac{\text{Transaction Send Rate}}{\text{Total time to execute } n \text{ transactions}} * \text{predictions per sec}$$

$$Throughput_{RAT} = \frac{\text{Transaction Send Rate}}{\text{Total time to execute } n \text{ transactions}}$$

Κεφάλαιο 6

Πειραματική Αποτίμηση

6.1 Προσέγγιση RAT	64
6.1.1 Εισαγωγή δεδομένων	64
6.1.2 Υποβολή ερωτήσεων	65
6.2 Προσέγγιση MAT.....	66
6.2.1 Εισαγωγή δεδομένων	66
6.2.2 Υποβολή ερωτήσεων	68
6.3 Σύγκριση RAT και MAT	68

Το παρόν κεφάλαιο αφορά την αποτίμηση των πειραμάτων όπως αυτά ορίστηκαν στο προηγούμενο κεφάλαιο. Αρχικά, θα σχολιάσουμε τα αποτελέσματα για τις προσεγγίσεις RAT και MAT ξεχωριστά. Ειδικότερα, υπάρχουν τρεις σειρές πειραμάτων, μια που αφορά την εισαγωγή δεδομένων και υποβολή ερωτήσεων, μια που αφορά *Throughput* και *Latency* και τέλος μια που αφορά την χρήση *CouchDB* ή *LevelDB* ως *World State*.

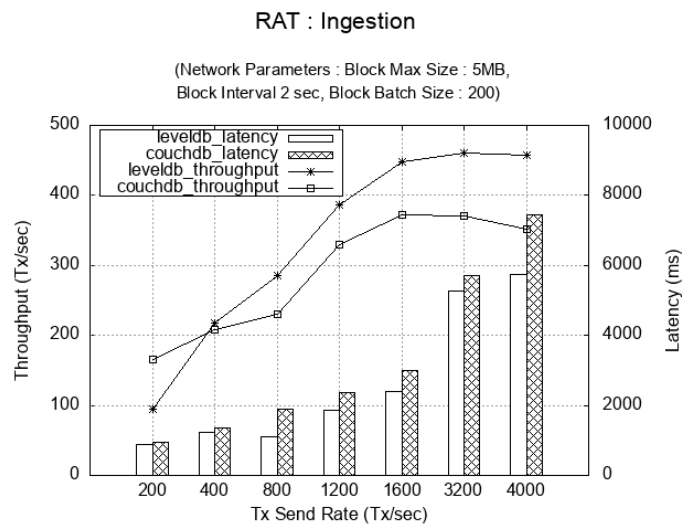
Ακόμη, σε αυτό το κεφάλαιο, θα δούμε την απόδοση του συστήματος σε συγκεντρωτικές γραφικές παραστάσεις, οι οποίες θα μας βοηθήσουν να εξάγουμε συμπεράσματα, τα οποία θα αναλύσουμε στο τελευταίο μέρος αυτού του κεφαλαίου.

Τέλος, αξιοσημείωτο είναι το γεγονός ότι η αρχιτεκτονική *TriaBase* έχει πολύ καλές επιδόσεις όσον αφορά την υποβολή ερωτήσεων, αφού ξεπερνά κατά χιλιάδες την κλασική προσέγγιση RAT.

6.1 Προσέγγιση RAT

Η προσέγγιση RAT, όπως αυτή περιεγράφηκε σε προηγούμενο κεφάλαιο, αφορά την απλή εισαγωγή δεδομένων σε ένα σύστημα αλυσίδας συστοιχιών (*Blockchain*) όπου κάθε δοσοληψία (transaction) αντιπροσωπεύει μια εγγραφή *Telco*. Πιο κάτω βλέπουμε τις γραφικές παραστάσεις για τα πειράματα εισαγωγής δεδομένων και υποβολής επερωτήσεων.

6.1.1 Εισαγωγή δεδομένων



Σχήμα 6.1 Αναπαράσταση *Throughput* και *Latency* για την εισαγωγή δεδομένων στην προσέγγιση RAT

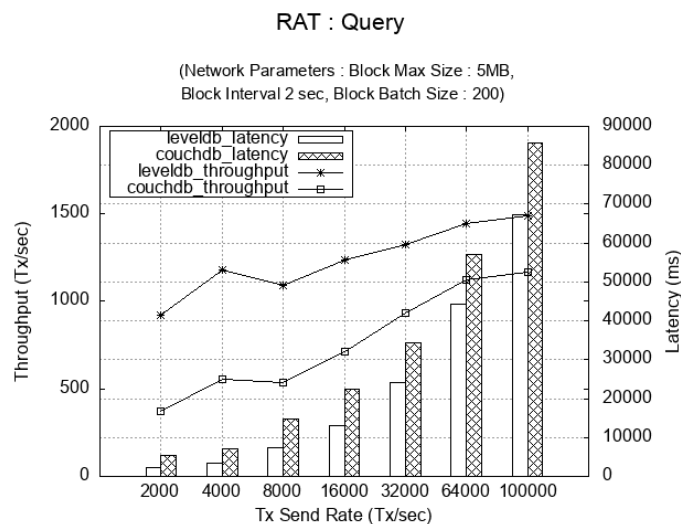
Στην πιο πάνω γραφική παράσταση βλέπουμε την απόδοση του τμήματος αλυσίδας συστοιχιών (*Blockchain Division*) όσον αφορά το *Throughput* και το και το *Latency*, για διαφορετικούς ρυθμούς υποβολής *transactions*. Ταυτόχρονα, στην γραφική παράσταση μπορούμε να δούμε τις ίδιες μετρήσεις, για διαφορετικά *World State*. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, το *Hyperledger Fabric* υποστηρίζει την λειτουργία *CouchDB* και *LevelDB*.

Αρχικά, στον πρωτεύον άξονα y μπορούμε να δούμε τον αριθμό των *transaction* που μπορεί να αφομοιώσει το σύστημα, ανά δευτερόλεπτο, για διάφορους ρυθμούς αποστολής *transactions*, όπως αυτοί φαίνονται στον άξονα x. Όσον αφορά την *LevelDB*,

παρατηρούμε, ότι το *Throughput* αυξάνεται σταδιακά από 100 Tx/Sec σε περίπου 450 Tx/Sec, όπου υπάρχει μια σταθεροποίηση και στην συνέχεια το *Throughput* τείνει να μειώνεται ελαφρώς. Παρόμοια, τάση βλέπουμε και στις μετρήσεις που αφορούν την *CouchDB* με 170 Tx/Sec έως 380 Tx/Sec αντίστοιχα. Από αυτό το σημείο και μετά παρατηρούμε μια σταδιακή ύφεση. Ακόμη, είναι οφθαλμοφανές ότι η *CouchDB* έχει χειρότερες επιδόσεις από την *LevelDB*.

Στον δευτεύον άξονα y μπορούμε να δούμε την καθυστέρηση *Latency* για τις δύο περιπτώσεις. Αρχικά, παρατηρούμε ότι για μεγαλύτερα Send Rate το *Latency* αυξάνεται εκθετικά και για τα δύο *World State*. Συγκεκριμένα για την *LevelDB*, υπάρχει μια άνοδος, από περίπου 1000 ms σε σχεδόν 6000 ms για Send Rate 200 Tx/Sec και 4000 Tx/Sec αντίστοιχα. Παρομοίως για την *CouchDB*, το *Latency* αυξάνεται από περίπου 1000 ms σε περίπου 7000 ms για Send Rate 200 Tx/Sec και 4000 Tx/Sec. Τέλος, μελετώντας το *Latency*, είναι πασιφανές ότι η *LevelDB* υπερτερεί της *CouchDB*.

6.1.2 Υποβολή επερωτήσεων



Σχήμα 6.2 Αναπαράσταση *Throughput* και *Latency* για την υποβολή επερωτήσεων στην προσέγγιση RAT

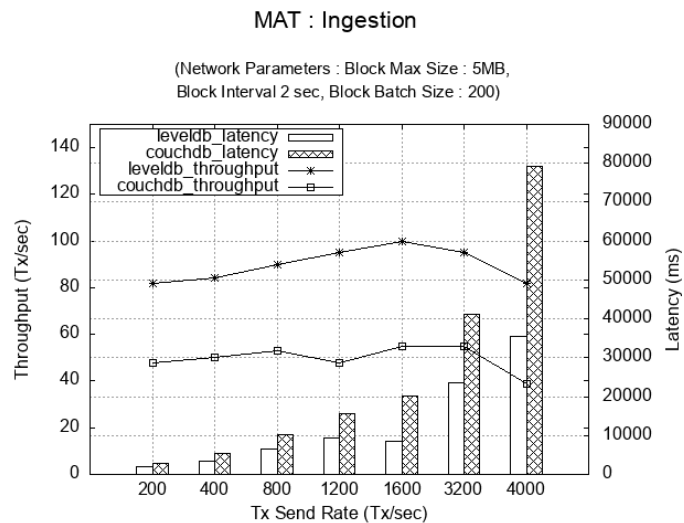
Στην πιο πάνω γραφική παράσταση βλέπουμε τα αντίστοιχα πειράματα που έγιναν για την υποβολή επερωτήσεων. Αναλυτικότερα, όσον αφορά το *Throughput*, με την χρήση *LevelDB*, αυτό αυξάνεται από 900 Tx/Sec σε 1500 Tx/Sec, για Send Rate 200 Tx/Sec

και 100000 Tx/Sec αντίστοιχα. Όσον αφορά την CouchDB, αυτή παρουσιάζει χειρότερες επιδόσεις, με αντίστοιχες μετρήσεις από 400 Tx/Sec μέχρι 1100 Tx/Sec. Στη συνέχεια, το *Latency* αυξάνεται εκθετικά και για τα δύο *World State Databases*. Συγκεκριμένα, για την περίπτωση της *LevelDB*, οι τιμές αυξάνονται εκθετικά από 2000 ms σε 67000 ms , ενώ για την περίπτωση της *CouchDB* από 5400 ms σε 84000 ms αντίστοιχα. Το γεγονός αυτό οφείλεται στο ότι τα *World States* δυσκολεύονται να εξυπηρετήσουν μεγάλο αριθμό επερωτήσεων. Τέλος, σε κάθε περίπτωση ή *LevelDB* υπερτερεί έναντι της *CouchDB*.

6.2 Προσέγγιση MAT

Η προσέγγιση MAT, όπως αυτή περιεγράφηκε σε προηγούμενο κεφάλαιο, αφορά την απλή εισαγωγή δεδομένων σε ένα σύστημα *Blockchain* όπου κάθε δοσοληψία αντιπροσωπεύει ένα μοντέλο *Machine learning* το οποίο εκπαιδεύτηκε με 1000 *Telco* εγγραφές. Πιο κάτω βλέπουμε τις γραφικές παραστάσεις για τα πειράματα εισαγωγής δεδομένων και υποβολής επερωτήσεων σε αυτό.

6.2.1 Εισαγωγή δεδομένων

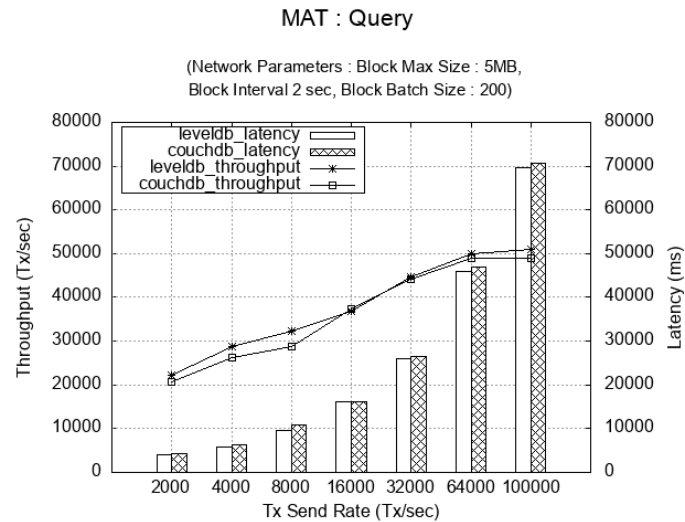


Σχήμα 6.3 Αναπαράσταση *Throughput* και *Latency* για την εισαγωγή δεδομένων στην προσέγγιση MAT

Αρχικά, στον πρωτεύον άξονα y μπορούμε να δούμε τον αριθμό των *transaction* που μπορεί να αφομοιώσει το σύστημα, ανά δευτερόλεπτο, για διάφορους ρυθμούς αποστολής *transactions*, όπως αυτοί φαίνονται στον άξονα x. Όσον αφορά την *LevelDB*, παρατηρούμε, ότι το *Throughput* αυξάνεται με μικρό ρυθμό, από 80 Tx/Sec σε περίπου 100 Tx/Sec, όπου υπάρχει μια ύφεση. Παρόμοια, τάση βλέπουμε και στις μετρήσεις που αφορούν την *CouchDB* με 50 Tx/Sec έως 60 Tx/Sec αντίστοιχα, όπου στην συνέχεια υπάρχει μια ύφεση. Το γεγονός αυτό οφείλεται κυρίως στην αδυναμία των *World State* να αφομοιώσουν με εφόλκια εγγραφές μεγάλου μεγέθους. Ακόμη, είναι οφθαλμοφανές ότι η *LevelDB* μπορεί να χειριστεί σχεδόν τα διπλάσια *transaction* σε σχέση με την *LevelDB*. Δεδομένου ότι τα μοντέλα που δίνονται ως είσοδος στις δοκιμές είναι της τάξης μεγέθους των 80 KB, μπορούμε να πούμε ότι η *LevelDB* χειρίζεται πιο αποδοτικά δεδομένα μεγαλύτερου όγκου.

Στον δευτερεύον άξονα y μπορούμε να δούμε την καθυστέρηση *Latency* για τις δύο περιπτώσεις. Αρχικά, παρατηρούμε ότι για μεγαλύτερα Send Rate, το *Latency* αυξάνεται εκθετικά και για τα δύο *World State*. Συγκεκριμένα για την *LevelDB* υπάρχει μια άνοδος, από περίπου 1000 ms σε σχεδόν 35000 ms για Send Rate 200 Tx/Sec και 4000 Tx/Sec αντίστοιχα. Παρομοίως για την *CouchDB* το *Latency* αυξάνεται από περίπου 1000 ms σε περίπου 80000 ms, για Send Rate 200 Tx/Sec και 4000 Tx/Sec. Ακόμη, συγκρίνοντας τα *Latency* μεταξύ *CouchDB* και *LevelDB* μπορούμε να πούμε γενικά, ότι η *CouchDB* χρειάζεται τον διπλάσιο χρόνο για να επικυρώσει (να κάνει *commit*) ένα μοντέλο σε σχέση με την *LevelDB*.

6.2.2 Υποβολή ερωτήσεων



Σχήμα 6.4 Αναπαράσταση *Throughput* και *Latency* για την υποβολή ερωτήσεων στην προσέγγιση MAT

Στην πιο πάνω γραφική παράσταση βλέπουμε τα αντίστοιχα πειράματα που έγιναν για την υποβολή ερωτήσεων. Αναλυτικότερα, όσον αφορά το *Throughput*, παρατηρούμε ότι οι δύο γραμμές που αφορούν τα *World States*, είναι στενά συσχετισμένες και σχεδόν συμβαδίζουν. Συγκεκριμένα παρατηρούμε μια σταθερή αύξηση του *Throughput* από 20000 Tx/Sec σε 50000 Tx/Sec. Στη συνέχεια, το *Latency* αυξάνεται εκθετικά και για τα δύο *World State Databases*, όπου οι τιμές αυξάνονται από 2000 ms σε 67000 ms. Επίσης, παρατηρούμε ότι το *Latency* συμβαδίζει για τις δύο περιπτώσεις.

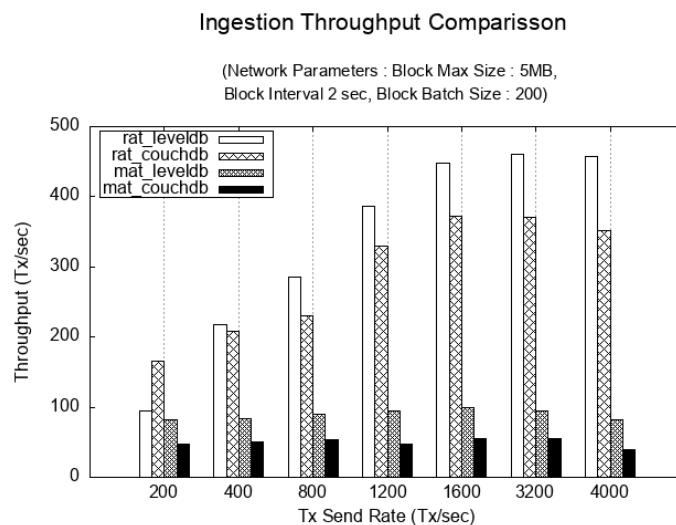
6.3 Σύγκριση RAT και MAT

Λαμβάνοντας υπόψη τις μετρήσεις που έγιναν στις πιο πάνω γραφικές παραστάσεις δημιουργήθηκαν οι δύο πιο κάτω συγκεντρωτικές γραφικές παραστάσεις οι οποίες έχουν σκοπό να αναδείξουν τις κύριες διαφορές μεταξύ των προσεγγίσεων RAT και MAT. Υπενθυμίζουμε ότι η προσέγγιση RAT χρησιμοποιείται από κοινές αρχιτεκτονικές *Blockchain*, ενώ το MAT είναι μια προσέγγιση την οποία προτείνουμε μέσα από την αρχιτεκτονική *TriaBase*.

Ingestion:

Αρχικά θα συγκρίνουμε την απόδοση όσον αφορά την εισαγωγή δεδομένων. Η προσέγγιση RAT φαίνεται να έχει μεγαλύτερη επιτυχία στην εισαγωγή δεδομένων, με επιδόσεις σχεδόν πέντε φορές καλύτερες από την προσέγγιση MAT. Το γεγονός αυτό, οφείλεται στο μέγεθος των *transaction* της κάθε προσέγγισης. Είναι γνωστό ότι μια *Telco* εγγραφή, έχει μέγεθος 75 byte ενώ το αντίστοιχο μοντέλο έχει μέγεθος περίπου 75000 bytes. Συνεπώς, οι χαμηλές επιδόσεις στην προσέγγιση MAT ήταν αναμενόμενες, αφού τα *World States* δεν είναι ικανά να χειριστούν με μεγάλη ευκολία objects μεγάλου μεγέθους.

Πέραν τούτου, σε κάθε περίπτωση μπορούμε να δούμε ότι η *CouchDB* έχει χειρότερη επίδοση σε σχέση με την *LevelDB*. Ακόμη, φαίνεται ότι η *LevelDB* μπορεί να χειριστεί με μεγαλύτερη ευκολία τον αυξανόμενο ρυθμό υποβολής *transactions*.



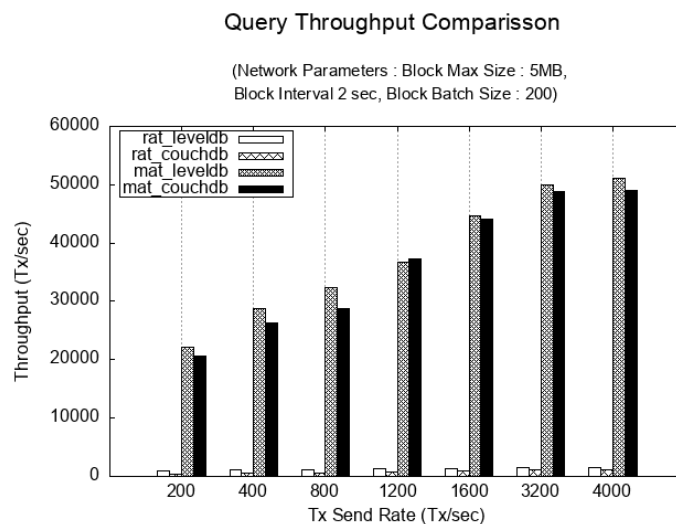
Σχήμα 6.5 Σύγκριση RAT και PAT όσον αφορά το Throughput εισαγωγής δεδομένων

Query:

Σε αυτή την παράγραφο θα δούμε την επίδοση του συστήματος όσον αφορά την υποβολή επερωτήσεων. Η προσέγγιση MAT φαίνεται να έχει μεγαλύτερη επιτυχία στην υποβολή επερωτήσεων, με *Throughput* που ξεπερνά κατά χιλιάδες *transaction* ανά δευτερόλεπτο, την προσέγγιση RAT. Το γεγονός αυτό οφείλεται στο ότι ένα μοντέλο *Machine learning*

μπορεί να εξυπηρετήσει χιλιάδες επερωτήσεις (queries), κάνοντας προβλέψεις για την τιμή μιας συγκεκριμένης εγγραφής σε ελάχιστο χρόνο .

Πέραν τούτου, σε κάθε περίπτωση μπορούμε να δούμε ότι η χρήση διαφορετικού *World State Database* δεν διαδραματίζει σημαντικό ρόλο στην απόδοση του συστήματος, αφού η διαδικασία υποβολής επερωτήσεων λαμβάνει μέρος σε μεγάλο βαθμό, στο μοντέλο *Machine learning*. Παρόλα αυτά για σκοπούς σύγκρισης είμαστε υπόχρεοι να αναφέρουμε ότι στις πλείστες των περιπτώσεων η *LevelDB* είναι ελαφρώς καλύτερη από την *CouchDB*.



Σχήμα 6.6 Σύγκριση RAT και PAT όσον αφορά το Throughput υποβολής επερωτήσεων

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα	71
7.2 Μελλοντική δουλειά	73

7.1 Συμπεράσματα

Ανακεφαλαιώνοντας, αυτή η διπλωματική εργασία είχε σκοπό την μελέτη διάφορων τεχνολογιών αλυσίδας συστοιχιών (*Blockchain*) και την πρόταση μιας νέας αρχιτεκτονικής, η οποία στοχεύει στην αποδοτική και ασφαλή διαχείριση *IoT* δεδομένων. Αρχικά, μελετήθηκαν διάφορες τεχνολογίες *Blockchain* έτσι ώστε να προσανατολιστούμε καλύτερα στο πεδίο αυτό. Έπειτα, έγινε εκτεταμένη μελέτη και εκμάθηση ενός ιδιωτικού με άδειες (*Private Permissioned Blockchain*) εργαλείου, του *Hyperledger Fabric*. Στη συνέχεια, είδαμε διάφορες τεχνικές διαχείρισης *IoT* δεδομένων με την χρήση *NodeRed* και *Influxdb*. Ακόμη μελετήθηκαν τεχνικές αποθήκευσης δεδομένων με την χρήση μοντέλων μηχανικής μάθησης (*Machine learning*) *LSTM*, καθώς επίσης και η τεχνική ομοσπονδιακής μάθησης (*Federated Learning*).

Όλοι οι στόχοι που θέσαμε αρχικά επιτεύχθηκαν με επιτυχία αφού το απότοκο της παρούσας δουλειάς, ήταν η διατύπωση και υλοποίηση της πρώτης έκδοσης της αρχιτεκτονικής *TriaBase*. Ειδικότερα, το πρωτότυπο που ετοιμάστηκε αφορά το σενάριο χρήσης της στον τομέα των τηλεπικοινωνιών. Η αρχιτεκτονική *TriaBase* είναι μοναδική και καινοτόμα αφού συνδυάζει την αποκεντρωμένη εκπαίδευση μοντέλων μηχανικής μάθησης στα άκρα του δικτύου, και την αποθήκευση τους με την χρήση της τεχνολογίας *Blockchain* σε ένα κατανεμημένο δίκτυο.

Η βασική δομή της *TriaBase* απαρτίζεται από το επίπεδο άκρων (*Edge Layer*), το επίπεδο εφαρμογής (*Application Layer*) και το επίπεδο αλυσίδας συστοιχιών (*Blockchain Network*). Στο *Edge Layer* μπορούμε να βρούμε τις διάφορες *IoT* συσκευές που παράγουν συνεχώς δεδομένα και τους *ML Client*, οι οποίοι εκπαιδεύουν τοπικά μοντέλα βάσει της πληροφορίας που παράγεται στα άκρα. Στο *Application Layer* έχουμε τους κόμβους της *TriaBase* οι οποίοι απαρτίζονται από δύο μέρη, το τμήμα χρήστη (*User Division*) και το τμήμα αλυσίδας συστοιχιών (*Blockchain Division*). Στο *User Division* μπορεί να συνδεθεί κάποιος εξουσιοδοτημένος χρήστης και να αρχικοποιήσει διεργασίες που αφορούν την εκπαίδευση μοντέλων, την αποθήκευσή τους στο *Blockchain* και την υποβολή επερωτήσεων σε αυτά. Επίσης, το *User Division*, αναλαμβάνει τον ρόλο του «Server» στην τεχνική *Federated Learning* και δημιουργεί ένα συγκεντρωτικό μοντέλο λαμβάνοντας υπόψη του τα τοπικά μοντέλα που έχουν παράξει οι *Clients*. Τέλος το *User Division* προωθεί τα συγκεντρωτικά μοντέλα στο *Blockchain Division* για αποθήκευση. Το *Blockchain Division* με την σειρά του ενεργοποιεί τον αλγόριθμο ομοφωνίας (*consensus*) του *Blockchain Network*, έτσι ώστε να διαδοθεί το μοντέλο στους ομότιμους κόμβους της *TriaBase* και να γίνει εν τέλει η επικύρωση (*commit*) στο καθολικό (*Ledger*). Σημαντικό είναι να αναφέρουμε ότι στο *Blockchain Division* φιλοξενούνται τα έξυπνα συμβόλαια (*smart contracts*), που αφορούν την αποθήκευση και ανάκτηση των μοντέλων. Η υποβολή επερωτήσεων στην *TriaBase* συμβαίνει μέσω του *User Division*, το οποίο ανακτά ένα τοπικό μοντέλο από το *Blockchain Division* και στην συνέχεια υποβάλλει σε αυτό διάφορες επερωτήσεις. Τέλος, το *Blockchain Network* δομείται από το *Hyperledger Fabric*.

Για την υλοποίηση της *TriaBase*, χρησιμοποιήθηκαν πολλές τεχνολογίες και γλώσσες προγραμματισμού. Αρχικά, για να αυτοματοποιήσουμε την δημιουργία του *Blockchain Network* έχουν γραφτεί *Shell Scripts*, ενώ ταυτόχρονα οι κόμβοι του *Hyperledger Fabric* έχουν στηθεί με την χρήση *Docker Compose* αρχείων. Επιπλέον, γράφτηκαν *smart contracts* για την υλοποίηση του *Blockchain Division*. Οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν για αυτό τον σκοπό είναι η *NodeJS* και η *Go*. Ακόμη, για την δημιουργία του *User Division*, έγινε χρήση της *NodeJS*. Όσον αφορά την εκπαίδευση των μοντέλων *Machine learning* χρησιμοποιήθηκε η *Python*.

Αξιολογώντας τα αποτελέσματα των πειραμάτων που έγιναν για την αποτίμηση της *TriaBase*, αυτά είναι θετικά. Ονομαστικά, υπάρχει ένας σταθερό *Throughput* όσον αφορά

την εισαγωγή δεδομένων, ενώ η υποβολή επρωτήσεων παρουσιάζει απίστευτα καλά αποτελέσματα. Το *Throughput* υποβολής επρωτήσεων στην *TriaBase* ανέρχεται σε περίπου 60000 στην καλύτερη περίπτωση, γεγονός που ξεπερνά το αμελητέο *Throughput* (1500 Tx/Sec), που βρίσκουμε στην απλή περίπτωση χρήσης του *Blockchain* για αποθήκευση της κάθε εγγραφής ξεχωριστά.

Σχετικά με τις προκλήσεις που αντιμετωπίσαμε κατά την διάρκεια εκπόνησης της πτυχιακής εργασίας, αυτές αφορούν κυρίως την εξοικείωση με τις τεχνολογίες *Blockchain*. Αναλυτικότερα, η εκμάθηση του εργαλείου *Hyperledger Fabric* ήταν αρκετά χρονοβόρα και η καμπύλη μάθησής του ήταν αρκετά μεγάλη. Ένα από τα κύρια προβλήματα που έπρεπε να αντιμετωπίσουμε ήταν η μεταβίβαση από την έκδοση 1.4 του *Hyperledger Fabric* στην έκδοση 2.0, όπου προστέθηκαν νέες δυνατότητες και ο κορμός λειτουργίας του *Framework* είχε αλλάξει ριζικά. Το γεγονός αυτό μας καθυστέρησε αρκετά αφού έπρεπε να διαβάσουμε αμέτρητες σελίδες από τον οδηγό χρήσης του εργαλείου και να βρούμε λύσεις σε διάφορα Bugs που προκύπταν με την χρήση της νέας έκδοσης. Τέλος, ο συνδυασμός της μηχανικής μάθησης με το *Blockchain* αποτέλεσε ακόμα ένα εμπόδιο στην υλοποίησή μας αφού δεν υπήρχε συμβατότητα μεταξύ των βιβλιοθηκών μηχανικής μάθησης και του *Hyperledger Fabric*. Παρόλα αυτά αντιμετωπίσαμε το πρόβλημα αυτό με την χρήση *RPC* πρωτοκόλλων.

7.2 Μελλοντική δουλειά

Σίγουρα, η δουλειά που έχει γίνει είναι αρκετή για να θέσει τα θεμέλια στον χώρο έρευνας των *Blockchain* για *IoT* δεδομένα με την χρήση *Machine learning*. Εντούτοις, οι βελτιώσεις που μπορούν να γίνουν είναι πολλές και απεριόριστες.

Στο μέλλον μπορούμε να επεκτείνουμε την προτεινόμενη αρχιτεκτονική, έτσι ώστε να εφαρμόζεται πιο εύκολα σε διάφορους τομείς. Μια περίπτωση που χρήζει εξερεύνησης είναι τα κλειστά συστήματα παρακολούθησης. Αναλυτικότερα, μπορούμε να εκπαιδεύουμε μοντέλα έτσι ώστε να αναγνωρίζουν συγκεκριμένες εικόνες οι οποίες λήφθηκαν από τις κάμερες. Τα μοντέλα αυτά θα αποθηκεύονται στο *Blockchain* έτσι ώστε να υπάρχει διαφάνεια και εμπιστοσύνη σε ένα δίκτυο με πολλούς εταίρους.

Επίσης, η χρήση του *Framework Hyperledger Fabric* μπορεί να αντικατασταθεί με ένα ειδικά διαμορφωμένο *Blockchain*, που να αρμόζει καλύτερα στην περίπτωση των *IoT* συσκευών. Επιπρόσθετα, σχετικά με το *Blockchain Network*, μπορούμε να εισάγουμε κάποιο σύστημα υπόληψης (reputation), όπου κάθε κόμβος στις *TriaBase* θα ανταμείβεται ανάλογα με την συνεισφορά του στο δίκτυο. Με αυτό τον τρόπο θα διασφαλίζεται η προστασία του δικτύου από πιθανούς κακόβουλους πράκτορες.

Τελειώνοντας, ένα άλλο μέρος της αρχιτεκτονικής *TriaBase* το οποίο χρήζει μελέτης, αφορά την τεχνολογία *Federated Learning*. Η εύρεση των βέλτιστων μοντέλων σε κάθε περίπτωση είναι μια δύσκολη εργασία. Η δημιουργία αποτελεσματικών μοντέλων τα οποία έχουν μικρό μέγεθος ενώ ταυτόχρονα εσωκλείουν μεγάλο όγκο πληροφορίας, είναι σημαντικό για κάθε *Blockchain*, αφού ένα μικρό μοντέλο επιφέρει σε λιγότερη συμφόρηση του δικτύου και συνεπώς στην καλύτερη απόδοση αυτού, όσον αφορά το *Throughput* και το *Latency*.

Αναφορές

- [1] "Wikipedia - *Bitcoin*," [Online]. Available: <https://en.wikipedia.org/wiki/Bitcoin>. [Accessed 2020].
- [2] "*Bitcoin* Home Page," [Online]. Available: <https://Bitcoin.org/el/>. [Accessed 2020].
- [3] S. Nakamoto, "*Bitcoin: A Peer-to-Peer Electronic Cash System*," 2008. [Online]. Available: <https://Bitcoin.org/Bitcoin.pdf>. [Accessed 2020].
- [4] "Public and Private *Blockchains*," [Online]. Available: <https://101Blockchains.com/public-vs-private-Blockchain/>. [Accessed 2020].
- [5] C. Mohan, "Tutorial: *Blockchains* and Databases," *VLDB Endowment*, vol. 10, no. 12, p. 2000–2001, 2017.
- [6] D. Massessi, "Public Vs Private *Blockchain* In A Nutshell," 12 12 2018. [Online]. Available: <https://medium.com/coinmonks/public-vs-private-Blockchain-in-a-nutshell-c9fe284fa39f>. [Accessed 2020].
- [7] "Consensus algorithms," [Online]. Available: <https://www.geeksforgeeks.org/consensus-algorithms-in-Blockchain/>. [Accessed 2020].
- [8] J. Sedlmeir, H. U. Buhl, G. Fridgen and R. Keller, "The Energy Consumption of *Blockchain* Technology: Beyond Myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599-608, 2020.
- [9] "Ethereum," [Online]. Available: <https://ethereum.org/en/>. [Accessed 2020].
- [10] "Wikipedia - Internet of things," [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things. [Accessed 2020].
- [11] F. Nawab, D. J. Abadi, O. Arden and M. Shadmon, "AnyLog: a Grand Unification of the Internet of Things," in *9th Biennial Conference on Innovative Data Systems Research*, California, USA, 2019.
- [12] M. J. Amiri, D. Agrawal and A. . E. Abbadi, "CAPER: a cross-application permissioned *Blockchain*," *VLDB Endowment*, vol. 12, no. 11, p. 1385–1398, 2019.
- [13] "Directed acyclic graph," [Online]. Available: https://en.wikipedia.org/wiki/Directed_acyclic_graph. [Accessed 2020].

- [14] "Influxdata," [Online]. Available: <https://docs.influxdata.com/Influxdb/cloud/reference/key-concepts/>. [Accessed 2020].
- [15] D. Zeinalipour, "Lecture 10," in *EPL646 – Advanced Topics in Databases*, 2020, p. 24.
- [16] J. Calusinski, "Build the foundation of an IoT app with Node-RED and Raspberry Pi," IBM, [Online]. Available: <https://www.ibm.com/cloud/architecture/tutorials/raspberry-pi-IoT/>. [Accessed 2020].
- [17] "Node-Red," IBM, [Online]. Available: <https://nodered.org/>. [Accessed 2020].
- [18] C. Costa, A. Charalambous, A. Konstantinides, D. Zeinalipour-Yazti and M. F. Mokbel, "Decaying Telco Big Data with Data Postdiction," in *Proceedings of the 19th IEEE International Conference on Mobile Data Management*, Aalborg, Denmark, 2018.
- [19] "Long short-term memory," [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory. [Accessed 2020].
- [20] B. McMahan and D. Ramage, "*Federated Learning: Collaborative Machine learning without Centralized Training Data*," Google, 6 4 2017. [Online]. Available: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>. [Accessed 2020].
- [21] "*Federated Learning*," Google, [Online]. Available: <https://federated.withgoogle.com/>. [Accessed 2020].
- [22] "Wikipedia - *Peer-to-Peer*," [Online]. Available: <https://en.wikipedia.org/wiki/Peer-to-Peer>. [Accessed 2020].
- [23] "Tech Terms - P2P Definition," [Online]. Available: <https://techterms.com/definition/p2p>. [Accessed 2020].
- [24] "Wikipedia BitTorrent," [Online]. Available: <https://en.wikipedia.org/wiki/BitTorrent>. [Accessed 2020].
- [25] C. P. a. J. Pelzl, *Understanding Cryptography A Textbook for Students and Practitioners*, Springer Publishing Company, 2010.
- [26] "Wikipedia - Seasar Cipher," [Online]. Available: https://en.wikipedia.org/wiki/Caesar_cipher. [Accessed 2020].
- [27] E. Athanasopoulos, "Asymmetric Encryption and RSA," in *EPL326 Lecture Notes*, 2019, pp. 18-21.

- [28] E. Athanasopoulos, "Introduction to Systems Security," in *EPL326 Lecture Notes*, 2019, p. 27.
- [29] E. Athanasopoulos, "Asymmetric Encryption and RSA," in *EPL326 Lecture Notes*, 2019, pp. 12-17.
- [30] E. Athanasopoulos, "Cryptographic Hash Functions," in *EPL326 Lecture Notes*, 2019, pp. 11-16.
- [31] E. Athanasopoulos, "Cryptographic Hash Functions," in *EPL 326 – Lecture Notes*, 2019, pp. 4-6.
- [32] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan and C. Murthy, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, Porto, Portugal, 2018.
- [33] "Hyperledger Fabric - A Blockchain Platform for the Enterprise," [Online]. Available: <https://Hyperledger-fabric.readthedocs.io/en/latest/>. [Accessed 2020].
- [34] "Smart contracts," [Online]. Available: <https://www.computerworld.com/article/3412140/whats-a-smart-contract-and-how-does-it-work.html>. [Accessed 2020].
- [35] "Solidity," [Online]. Available: <https://docs.soliditylang.org/en/v0.7.4/>. [Accessed 2020].
- [36] "The Raft Consensus Algorithm," [Online]. Available: <https://raft.github.io/>. [Accessed 2020].
- [37] "Web 3.0: the decentralised web promises to make the internet free again," 11 March 2019. [Online]. Available: <https://theconversation.com/web-3-0-the-decentralised-web-promises-to-make-the-internet-free-again-113139>. [Accessed 2020].
- [38] D. Zeinalipour, "Lecture 18," in *EPL646 – Advanced Topics in Databases*, 2020.
- [39] "Hyperledger Fabric SDK NodeJS," [Online]. Available: <https://Hyperledger.github.io/fabric-sdk-node/release-1.4/module-fabric-network.html>. [Accessed 2020].
- [40] Y. Polat, "Build a Blockchain Network for trusted IoT," IBM, 18 2019. [Online]. Available: <https://developer.ibm.com/technologies/Blockchain/patterns/build-a-Blockchain-network-for-trusted-IoT/>. [Accessed 2020].

Παράρτημα «Α»

Σε αυτό το παράρτημα υπάρχει ένας μικρός οδηγός χρήσης για το στήσιμο του *Hyperledger Fabric* βάση των Shell Scripts που υπάρχουν στο Παράρτημα «Β».

Οι ακόλουθες λειτουργίες αφορούν την χρήση του *Hyperledger Fabric* σε Linux μηχανή, Ubuntu 18.04.4 LTS.

Προαπαιτούμενα λειτουργίας Hyperledger Fabric

Εγκατάσταση Git

```
$ sudo apt install git
```

Εγκατάσταση cURL

```
$ sudo apt install curl
```

Εγκατάσταση wget

```
$ sudo apt install wget
```

Εγκατάσταση Docker

```
$ sudo apt install docker.io  
$ sudo systemctl start docker  
$ sudo systemctl enable docker
```

Εγκατάσταση Docker Compose

```
$ sudo curl -L  
"https://github.com/docker/compose/releases/download/1.26.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
$ sudo chmod +x /usr/local/bin/docker-compose
```

Εγκατάσταση Golang

```
$ wget https://dl.google.com/go/go1.13.8.linux-amd64.tar.gz  
$ sudo tar -xvf go1.13.8.linux-amd64.tar.gz  
$ sudo mv go /usr/local.
```

Οι πιο κάτω εντολές πρέπει να προστεθούν στο αντίστοιχο αρχείο profile του χρήστη.

```
export GOROOT=/usr/local/go  
export GOPATH=$HOME/go  
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

Μετά τρέχουμε την εντολή `$source` για να ενημερωθεί το terminal.

Εγκατάσταση NodeJS

```
$ curl -o- https://raw.githubusercontent.com/nvm-  
sh/nvm/v0.35.3/install.sh | bash
```

Μετά τρέχουμε την εντολή `$source` για να ενημερωθεί το terminal.

```
$ nvm install 8.9.4  
$ nvm install 12.19.0
```

Εγκατάσταση *Hyperledger Fabric* Docker Images

```
$ curl -sSL https://bit.ly/2ysb0FE | bash -s -- 2.0.1 1.4.6 0.4.18
```

Έγερση δικτύου *Hyperledger Fabric*

Κατεβάζουμε το repository που έχει στηθεί στο GitHub, το οποίο περιλαμβάνει τα σχετικά Shell Scripts και Configuration Files.

```
$ git clone https://github.com/Erodotos/Hyperledger-Fabric-Network.git
```

Αφού κατεβάσουμε το σχετικό repository προηγούμεστε στον φάκελο που έχει δημιουργηθεί. Εκεί βλέπουμε πολλούς υποφακέλους. Για να ξεκινήσουμε την διαδικασία, πάμε στον φάκελο `./network`. Στην συνέχεια εισάγουμε τις ακόλουθες εντολές και περιμένουμε υπομονετικά την εκτέλεσή τους αφού αυτές μπορεί να διαρκέσουν μερικά λεπτά.

```
$ network.sh up -c mychannel -db couchdb
```

```
$ network.sh deployCC -c mychannel -ccn contract_m1Model -ccv 1 -ccp  
../chaincode/src/m1Model/NodeJS/contract -ccl node
```

Μετά την εκτέλεση των πιο πάνω εντολών έχει στηθεί ένα δίκτυο *Hyperledger Fabric* ενώ ταυτόχρονα έχει εγκατασταθεί στους κόμβους του, το σχετικό *smart contract* για την εγγραφή και ανάκτηση *Machine learning* Μοντέλων στο *Blockchain*. Το δίκτυο μας αποτελείται από δύο οργανισμούς με δύο *Peers* ο καθένας, ενώ ως *World State* στην συγκεκριμένη περίπτωση έχει τεθεί η CouchDB. Για να εκλέξουμε ότι το δίκτυο λειτουργεί ορθά μπορούμε να τρέξουμε την ακόλουθη εντολή η οποία δοκιμάζει να εισάγει ένα μοντέλο στο *Blockchain*.

```
$ network.sh testCC -c mychannel -ccn contract_m1Model
```

Για να τερματίσουμε την λειτουργία του δικτύου τρέχουμε την ακόλουθη εντολή.

```
$ network.sh down
```

Για την εκτύπωση του οδηγού χρήσης στο terminal εισάγουμε την πιο κάτω εντολή.

```
$ network.sh -h
```

Βασική περίπτωση χρήσης του Bench Tool, για εισαγωγή δεδομένων

Αρχίζοντας από τον βασικό φάκελο του repository προηγούμεστε στον υποφάκελο `./app/bench-tool`. Αρχικά, θα πρέπει να έχουμε κάποια προκαθορισμένα credentials έτσι ώστε να έχουμε πρόσβαση στο *Blockchain*. Για να το κάνουμε αυτό τρέχουμε τις ακόλουθες εντολές.

```
$ nvm use 12.19.0
```

```
$ node credstore.js org1 Admin
```

```
$ node credstore.js org2 Admin
```

Στην συνέχεια τρέχουμε τον `manager.js` με τις κατάλληλες παραμέτρους. Παράδειγμα εντολής που θα δοθεί παρουσιάζεται πιο κάτω.

```
$ node manager.js MAT
```

Παράρτημα «B»

Σε αυτό το παράρτημα αναγράφεται ο κώδικας ο οποίος γράφτηκε. Στο πιο κάτω δένδρόγραμμα φαίνονται τα πιο σημαντικά αρχεία που συγγράφηκαν για τον σκοπό της παρούσας διπλωματικής εργασίας.

```
├─ app
  │├─ bench-tool
  │ │├─ cred-store.js
  │ │├─ manager.js
  │ │├─ manager_query.js
  │ │├─ profiles
  │ │ │├─ dev-connect.yaml
  │ │ │├─ org1-Client.yaml
  │ │ │├─ org2-Client.yaml
  │ │├─ testMAT.js
  │ │├─ testMAT_query.js
  │ │├─ testRAT.js
  │ │├─ testRAT_query.js
  │├─ chaincode
  │ │├─ src
  │ │ │├─ mlModel
  │ │ │ │├─ contract
  │ │ │ │ │├─ lib
  │ │ │ │ │├─ contract_mlModel.js
  │ │ │├─ rawOne
  │ │ │ │├─ contract_RAT.go
  │├─ network
  │ │├─ config
  │ │ │├─ configtx.yaml
  │ │ │├─ core.yaml
  │ │ │├─ crypto-config.yaml
  │ │├─ docker
  │ │ │├─ docker-compose-couch.yaml
  │ │ │├─ docker-compose.yaml
  │ │├─ network.sh
  │ │├─ scripts
  │ │ │├─ createChannel.sh
  │ │ │├─ deployCC.sh
  │ │ │├─ envVar.sh
  │ │ │├─ testChaincode_mlModel.sh
  │ │ │├─ testChaincode_rawOne.sh
  │ │├─ scriptUtils.sh
  │├─ Telco_mdm_keras.py
  │├─ LSTM_predictor_keras.py
```



```
1  /**
2   * Demonstrates the setup of the credential store
3   */
4  const fs = require('fs');
5  const Client = require('fabric-client');
6  const path = require('path');
7  const { exit } = require('process');
8
9
10 var myArgs = process.argv.slice(2);
11 const ORG_NAME = myArgs[0];
12
13 // Constants for profile, wallet & user
14 const CONNECTION_PROFILE_PATH = './profiles/dev-connect.yaml'
15
16 // Client section configuration
17 var CLIENT_CONNECTION_PROFILE_PATH;
18 if (ORG_NAME === 'org1') {
19   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org1-client.yaml';
20 } else if (ORG_NAME === 'org2') {
21   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org2-client.yaml';
22 } else {
23   console.log("Exiting : Invalid Organization name");
24   exit(0);
25 }
26
27
28 const CRYPTO_CONFIG = path.resolve(__dirname, '../network/organizations');
29 const CRYPTO_CONFIG_PEER_ORGANIZATIONS = path.join(CRYPTO_CONFIG, 'peerOrganizations')
30
31 const client = Client.loadFromConfig(CONNECTION_PROFILE_PATH)
32
33 main()
34
35 // Main function
36 async function main() {
37   // Input data sanity check
38   if (process.argv.length < 4) {
39     console.log("Usage: node cred-store.js org user-name")
40     process.exit(1)
41   }
42
43   // Set the org and name local variables
44   let org = process.argv[2]
45   let user = process.argv[3]
46
47   // Load the client section for the organization
48   // This has the location of the credential store
49
50   client.loadFromConfig(CLIENT_CONNECTION_PROFILE_PATH)
51
52
53   // Initialize the store
54   await initCredentialStore()
55
56   // Lets get the specified user from the store
57   let userContext = await client.loadUserFromStateStore(user)
58
59   // If user is null then the user does not exist
60   if (userContext == null) {
61
62     // Create the user context
```

Jan 09, 21 13:08	cred-store.js	Page 2/3
<pre> 63 userContext = await createUserContext(org, user) 64 65 console.log('Created \${user} under the credentials store!!') 66 } else { 67 console.log('User \${user} already exist!!') 68 } 69 70 // Setup the context on the client 71 await client.setUserContext(userContext, false) 72 73 // At this point client can be used for sending requests to Peer Orderer 74 } 75 76 async function initCredentialStore() { 77 78 // Call the function for initializing the credentials store on file system 79 await client.initCredentialStores() 80 .then((done) => { 81 console.log("initCredentialStore(): ", done) 82 }) 83 } 84 85 async function createUserContext(org, user) { 86 // Get the path to user private key 87 let privateKeyPath = getPrivateKeyPath(org, user) 88 89 // Get the path to the user certificate 90 let certPath = getCertPath(org, user) 91 92 // Setup the options for the user context 93 // Global Type: UserOpts 94 let opts = { 95 username: user, 96 mspid: createMSPId(org), 97 cryptoContent: { 98 privateKey: privateKeyPath, 99 signedCert: certPath 100 }, 101 skipPersistence: false 102 } 103 104 // Setup the user 105 let userContext = await client.createUser(opts) 106 107 return userContext 108 } 109 110 function getCertPath(org, user) { 111 var certPath = CRYPTO_CONFIG_PEER_ORGANIZATIONS + "/" + org + ".example.com/users/" + user + "@" + org + ".example.com/msp/signcerts/" + user + "@" + org + ".example.com-cert.pem" 112 return certPath 113 } 114 115 function getPrivateKeyPath(org, user) { 116 var pkFolder = CRYPTO_CONFIG_PEER_ORGANIZATIONS + "/" + org + ".example.com/users/" + user + "@" + org + ".example.com/msp/keystore" 117 fs.readdirSync(pkFolder).forEach(file => { 118 pkfile = file 119 return 120 }) 121 122 return (pkFolder + "/" + pkfile) </pre>		
Jan 09, 21 13:08	cred-store.js	Page 3/3
<pre> 123 } 124 125 function createMSPId(org) { 126 return org.charAt(0).toUpperCase() + org.slice(1) + 'MSP' 127 } </pre>		

```
1  'use strict';
2
3  const { Console } = require('console');
4  const { exit } = require('process');
5  const execFile = require('child_process').execFile;
6
7
8  var myArgs = process.argv.slice(2);
9  var approach = myArgs[0];
10
11 var hrstart = process.hrtime()
12
13 var organizations = ['org1', 'org2'];
14 var users = ['Admin'];
15 var peers = ['0', '1'];
16 var channel = 'mychannel';
17 var contract;
18 if (approach === 'RAT') {
19   contract = 'contract_rawOne';
20 } else if (approach === 'PAT') {
21   contract = 'contract_rawTwo';
22 } else if (approach === 'MAT') {
23   contract = 'contract_mlModel';
24 } else {
25   console.log('The given approach is not recognized');
26   exit(0);
27 }
28
29
30
31 let peerName;
32 let child;
33
34
35 console.log("\n\n\tSend Rate = 4000");
36 var transactionNumber = 1000; //1/4
37
38 for (var org of organizations) {
39   for (var peer of peers) {
40     peerName = 'peer' + peer + '.' + org + '.' + 'example.com';
41     child = execFile('node', ['test'+approach+'.js', org, users[0], peerName, c
42 hannel, contract, transactionNumber], (error, stdout, stderr) => {
43       if (error) {
44         throw error;
45       }
46       console.log(stdout);
47     });
48   }
49 }
```

```

1  'use strict';
2
3  const { exit } = require('process');
4  const execFile = require('child_process').execFile;
5
6
7  var myArgs = process.argv.slice(2);
8  var approach = myArgs[0];
9  var multipeer = myArgs[1];
10 var totaltransactionNumber = myArgs[2];
11
12 if (!totaltransactionNumber){
13   console.log('Number of transactions unspecified. ');
14   exit(0);
15 }
16
17 var peerNumber = 4;
18 var channel = 'mychannel';
19 var organizations;
20 var users;
21 var peers;
22 var transactionNumber;
23 if (multipeer === 'true'){
24   organizations = ['org1', 'org2'];
25   users = ['Admin'];
26   peers = ['0', '1'];
27   transactionNumber = totaltransactionNumber/peerNumber;
28 }else if (multipeer === 'false'){
29   organizations = ['org1'];
30   users = ['Admin'];
31   peers = ['0'];
32   transactionNumber = totaltransactionNumber;
33 }else{
34   console.log('Multipeer parameter not specified. ');
35   exit(0);
36 }
37
38
39 var contract;
40 if (approach === 'RAT') {
41   contract = 'contract_mnist_rat';
42 } else if (approach === 'MAT') {
43   contract = 'contract_mnist_mat';
44 } else {
45   console.log('The given approach is not recognized. ');
46   exit(0);
47 }
48
49 let peerName;
50 let child;
51
52 console.log("\n\nTotal Transaction Number:\t\t", totaltransactionNumber);
53
54 var sum_of_peer_execution_time = 0;
55 var iteration_count = 0;
56 for (var org of organizations) {
57   for (var peer of peers) {
58     peerName = 'peer' + peer + '.' + org + '.' + 'example.com';
59     child = execFile('node', ['test'+approach+'_query.js', org, users[0], peerName, channel, contract, transactionNumber], (error, stdout, stderr) => {
60       if (error) {
61         throw error;
62       }

```

```

63
64   let temp = stdout.toString().split(' ');
65   sum_of_peer_execution_time += parseFloat((temp[2]+temp[3]).replace(", ", "."));
66
67   iteration_count++;
68   if (iteration_count===peerNumber){
69     let execution_time = sum_of_peer_execution_time/peerNumber;
70     let throughput = totaltransactionNumber/execution_time;
71     console.log('\nTotal Execution Time (seconds):\t\t', execution_time);
72     console.log('\nThroughput (transactions/seconds):\t\t', throughput, '\n');
73   }
74 }
75 }

```

```

1  ---
2  #
3  # The network connection profile provides client applications the information ab
4  # out the target
5  # blockchain network that are necessary for the applications to interact with it
6  .
7  name: "Fabric"
8  #
9  # Describe what the target network is/does.
10 #
11 description: "A test enviornment setup"
12 #
13 # Schema version of the content. Used by the SDK to apply the corresponding pars
14 # ing rules.
15 version: "1.0"
16 #
17 #
18 # list of participating organizations in this network
19 #
20 organizations:
21   Org1:
22     mspid: Org1MSP
23     peers:
24       -peer0.org1.example.com
25       -peer1.org1.example.com
26
27   Org2:
28     mspid: Org2MSP
29     peers:
30       -peer0.org2.example.com
31       -peer1.org2.example.com
32
33 #
34 # [Optional]. But most apps would have this section so that channel objects can
35 # be constructed
36 # based on the content below. If an app is creating channels, then it likely wil
37 # l not need this
38 # section.
39 #
40 channels:
41   # name of the channel
42   mychannel:
43     # List of orderers designated by the application to use for transactions on
44     # this channel.
45     orderers:
46       - orderer.example.com
47       # - orderer1.example.com
48
49     # Required. list of peers from participating orgs
50     peers:
51       peer0.org1.example.com:
52         endorsingPeer: true
53       peer1.org1.example.com:
54         endorsingPeer: true
55       peer0.org2.example.com:
56         endorsingPeer: true
57       peer1.org2.example.com:
58         endorsingPeer: true
59
60 #

```

```

58 # List of orderers to send transaction and channel create/update requests to. For the time
59 # being only one orderer is needed.
60 #
61 orderers:
62   orderer.example.com:
63     url: grpcs://localhost:7050
64
65   # these are standard properties defined by the gRPC library
66   # they will be passed in as-is to gRPC client constructor
67   grpcOptions:
68     ssl-target-name-override: orderer.example.com
69   tlsCACerts:
70     path: ../../network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
71
72 #
73 # List of peers to send various requests to, including endorsement, query
74 # and event listener registration.
75 #
76 peers:
77   peer0.org1.example.com:
78     url: grpcs://localhost:7051
79     grpcOptions:
80       ssl-target-name-override: peer0.org1.example.com
81     tlsCACerts:
82       path: ../../network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp/tlscacerts/tlsca.org1.example.com-cert.pem
83   peer1.org1.example.com:
84     url: grpcs://localhost:8051
85     grpcOptions:
86       ssl-target-name-override: peer1.org1.example.com
87     tlsCACerts:
88       path: ../../network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp/tlscacerts/tlsca.org1.example.com-cert.pem
89
90   peer0.org2.example.com:
91     url: grpcs://localhost:9051
92     grpcOptions:
93       ssl-target-name-override: peer0.org2.example.com
94     tlsCACerts:
95       path: ../../network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/msp/tlscacerts/tlsca.org2.example.com-cert.pem
96   peer1.org2.example.com:
97     url: grpcs://localhost:10051
98     grpcOptions:
99       ssl-target-name-override: peer1.org2.example.com
100    tlsCACerts:
101      path: ../../network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/msp/tlscacerts/tlsca.org2.example.com-cert.pem

```

Jan 09, 21 13:11	org1-client.yaml	Page 1/1
<pre> 1 # Sample YAML for client config 2 name: "Org1 Client" 3 version: "1.0" 4 description: "Client for Org1" 5 6 # Client section describes the client 7 client: 8 # Org name 9 organization: Org1 10 # User context storage 11 credentialStore: 12 # State 13 path: "./credstore/org1/hfc-kvs" 14 # Public & Private key 15 cryptoStore: 16 path: "./credstore/org1/hfc-cvs" 17 18 #GRPC options 19 connection: 20 options: 21 grpc.keepalive_time_ms: 120000 </pre>		

Jan 09, 21 13:11	org2-client.yaml	Page 1/1
<pre> 1 # Sample YAML for client config 2 name: "Client Org2" 3 version: "1.0" 4 description: "Client for Org2" 5 6 # Client section describes the client 7 client: 8 # Org name 9 organization: Org2 10 # User context storage 11 credentialStore: 12 # State 13 path: "./credstore/org2/hfc-kvs" 14 # Public & Private key 15 cryptoStore: 16 path: "./credstore/org2/hfc-cvs" 17 18 #GRPC options 19 connection: 20 options: 21 grpc.keepalive_time_ms: 120000 </pre>		

```

1  'use strict';
2  const fs = require('fs');
3  const Client = require('fabric-client');
4  const { exit } = require('process');
5
6  var myArgs = process.argv.slice(2);
7
8  // Arguments Parsing
9  const ORG_NAME = myArgs[0];
10 const USER_NAME = myArgs[1];
11 const PEER_NAME = myArgs[2];
12 const CHANNEL_NAME = myArgs[3];
13 const CHAINCODE_ID = myArgs[4];
14 const NUMBER_OF_TXS = parseInt(myArgs[5]);
15
16
17 // Constants for profile
18 const CONNECTION_PROFILE_PATH = './profiles/dev-connect.yaml';
19
20 var CLIENT_CONNECTION_PROFILE_PATH;
21 // Client section configuration
22 if (ORG_NAME === 'org1') {
23   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org1-client.yaml';
24 } else if (ORG_NAME === 'org2') {
25   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org2-client.yaml';
26 } else {
27   console.log("Exiting : Invalid Organization name");
28   exit(0);
29 }
30
31 //global counter
32 var committedTxS = 0;
33
34 // Variable to hold the client
35 var client = {};
36 // Variable to hold the channel
37 var channel = {}
38
39 var initial_timer;
40 var hrstart = [];
41 var hrend = [];
42 var total_time = [];
43
44 main()
45
46 async function main() {
47
48   client = await setupClient();
49
50   channel = await setupChannel();
51
52   var TxArray = [];
53   var Tx = {};
54   for (var i = 0; i < NUMBER_OF_TXS; i++) {
55     var meas_info = (Math.floor(10000000 + Math.random() * 90000000)).toString()
56     var counter = (Math.floor(10000000 + Math.random() * 90000000)).toString()
57
58     var id = meas_info + '_' + counter;
59     var model_json_base64 = generateBase64String(4688);
60     var model_weights_base64 = generateBase64String(67760);
61     Tx.id = id;

```



```

62     Tx.model_json_base64 = model_json_base64;
63     Tx.model_weights_base64 = model_weights_base64;
64
65     TxArray.push(Tx);
66 }
67
68     initial_timer = process.hrtime();
69     for (var i = 0; i < NUMBER_OF_TXS; i++) {
70         invokeChaincode(TxArray[i]);
71     }
72 }
73
74
75 async function invokeChaincode(tx_data) {
76
77     let peerName = channel.getChannelPeer(PEER_NAME)
78
79     var tx_id = client.newTransactionID();
80     let tx_id_string = tx_id.getTransactionID();
81
82     var request = {
83         targets: peerName,
84         chaincodeId: CHAINCODE_ID,
85         fcn: 'writeModel',
86         args: [tx_data.id, tx_data.model_json_base64, tx_data.model_weights_base
64],
87         chainId: CHANNEL_NAME,
88         txId: tx_id
89     };
90
91     hrstart.push(process.hrtime());
92
93     // console.log("#1 channel.sendTransactionProposal Done.")
94     let results = await channel.sendTransactionProposal(request);
95
96     // Array of proposal responses
97     var proposalResponses = results[0];
98
99     var proposal = results[1];
100
101     var all_good = true;
102     for (var i in proposalResponses) {
103         let good = false
104         if (proposalResponses[i].response && proposalResponses[i].response.status === 200) {
105             good = true;
106             // console.log('\tinvoke chaincode EP response #{i} was good');
107         } else {
108             // console.log('\tinvoke chaincode EP response #{i} was bad!!!');
109         }
110         all_good = all_good & good
111     }
112     // console.log("#2 Looped through the EP results all_good=", all_good)
113
114     await setupTxListener(tx_id_string)
115     // console.log('#3 Registered the Tx Listener')
116
117     var orderer_request = {
118         txId: tx_id,
119         proposalResponses: proposalResponses,
120         proposal: proposal
121     };
122
123

```

```
185         console.log("User NOT found in credstore: ", USER_NAME)
186         process.exit(1)
187     }
188
189     client.setUserContext(userContext, true)
190
191     return client
192 }
193
194
195 async function setupChannel() {
196     try {
197         channel = await client.getChannel(CHANNEL_NAME, true)
198     } catch (e) {
199         console.log("Could NOT create channel: ", CHANNEL_NAME)
200         process.exit(1)
201     }
202
203     return channel
204 }
205
206 function generateBase64String(length) {
207     var result = '';
208     var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
209
210     var charactersLength = characters.length;
211     for (var i = 0; i < length; i++) {
212         result += characters.charAt(Math.floor(Math.random() * charactersLength))
213     }
214     return result;
215 }
```

```

124     await channel.sendTransaction(orderer_request);
125     // console.log("#4 channel.sendTransaction - waiting for Tx Event")
126
127 }
128
129
130 function setupTxListener(tx_id_string) {
131     let event_hub = channel.getChannelEventHub(PEER_NAME);
132
133     event_hub.registerTxEvent(tx_id_string, (tx, code, block_num) => {
134         // console.log("#5 Received Tx Event")
135         // console.log('\tThe chaincode invoke chaincode transaction has been co
136         mmitted on peer %s', event_hub.getPeerAddr());
137         // console.log('\tTransaction %s is in block %s', tx, block_num);
138
139         if (code !== 'VALID') {
140             // console.log('\tThe invoke chaincode transaction was invalid, code
141             :%s', code);
142         } else {
143             // console.log('\tThe invoke chaincode transaction was VALID.');
```

```
1  'use strict';
2  const fs = require('fs');
3  const Client = require('fabric-client');
4  const { exit } = require('process');
5
6  var myArgs = process.argv.slice(2);//get arguments from command line
7
8  // Arguments Parsing
9  const ORG_NAME = myArgs[0];
10 const USER_NAME = myArgs[1];
11 const PEER_NAME = myArgs[2];
12 const CHANNEL_NAME = myArgs[3];
13 const CHAINCODE_ID = myArgs[4];
14 const NUMBER_OF_TXS = parseInt(myArgs[5]);
15
16
17 // Constants for profile
18 const CONNECTION_PROFILE_PATH = './profiles/dev-connect.yaml';
19
20 var CLIENT_CONNECTION_PROFILE_PATH;
21 // Client section configuration
22 if (ORG_NAME === 'org1') {
23     CLIENT_CONNECTION_PROFILE_PATH = './profiles/org1-client.yaml';
24 } else if (ORG_NAME === 'org2') {
25     CLIENT_CONNECTION_PROFILE_PATH = './profiles/org2-client.yaml';
26 } else {
27     console.log("Exiting : Invalid Organization name");
28     exit(0);
29 }
30
31 //global counter
32 var numTxs = 0;
33
34 // Variable to hold the client
35 var client = {};
36 // Variable to hold the channel
37 var channel = {}
38
39 var initial_timer;
40 var hrstart = [];
41 var hrend = [];
42 var total_time = [];
43
44 main()
45
46 async function main() {
47     client = await setupClient();
48
49     channel = await setupChannel();
50
51     let id = 'id_1';
52
53     initial_timer = process.hrtime();
54     for (var i = 0; i < NUMBER_OF_TXS; i++) {
55         queryImage(id); //async call
56     }
57
58 }
59
60
61
62 async function queryImage(id) {
63
```

```

64     let peerName = channel.getChannelPeer(PEER_NAME)
65
66     let request = {
67         targets: peerName,
68         chaincodeId: CHAINCODE_ID,
69         fcn: 'queryModel',
70         args: [id]
71     };
72
73     // send the query proposal to the peer
74     hrstart.push(process.hrtime());
75     let response = await channel.queryByChaincode(request);
76     hrend.push(process.hrtime(hrstart[numTxs])); //diff of start and end time
77     numTxs++;
78     if (numTxs === NUMBER_OF_TXS) {
79         var final_timer = process.hrtime(initial_timer);
80         console.log("\t\t", final_timer);
81         for (var timer of hrend) {
82             total_time.push(timer[0] * 1000 + timer[1] / 1000000);
83         }
84
85         var sum = total_time.reduce((acc, c) => acc + c, 0);
86         var average = sum / NUMBER_OF_TXS;
87
88         //console.log("\t\tAverage Retrieval Latency = %d ms", average);
89         //console.log("\t\tQueried Txs = ", numTxs);
90         exit(0);
91     }
92
93     return
94 }
95
96 async function setupClient() {
97
98     const client = Client.loadFromConfig(CONNECTION_PROFILE_PATH)
99
100     client.loadFromConfig(CLIENT_CONNECTION_PROFILE_PATH)
101
102     await client.initCredentialStores()
103         .then((done) => {
104             // console.log("initCredentialStore(): ", done)
105         })
106
107     let userContext = await client.loadUserFromStateStore(USER_NAME)
108     if (userContext == null) {
109         console.log("User NOT found in credstore: ", USER_NAME)
110         process.exit(1)
111     }
112
113     client.setUserContext(userContext, true)
114
115     return client
116 }
117
118 async function setupChannel() {
119     try {
120         channel = await client.getChannel(CHANNEL_NAME, true)
121     } catch (e) {
122         console.log("Could NOT create channel: ", CHANNEL_NAME)
123         process.exit(1)
124     }
125
126     return channel

```

```

1  'use strict';
2  const fs = require('fs');
3  const Client = require('fabric-client');
4  const { exit } = require('process');
5
6  var myArgs = process.argv.slice(2);
7
8  // Arguments Parsing
9  const ORG_NAME = myArgs[0];
10 const USER_NAME = myArgs[1];
11 const PEER_NAME = myArgs[2];
12 const CHANNEL_NAME = myArgs[3];
13 const CHAINCODE_ID = myArgs[4];
14 const NUMBER_OF_TXS = parseInt(myArgs[5]);
15
16 // Constants for profile
17 const CONNECTION_PROFILE_PATH = './profiles/dev-connect.yaml';
18
19 var CLIENT_CONNECTION_PROFILE_PATH;
20 // Client section configuration
21 if (ORG_NAME === 'org1') {
22   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org1-client.yaml';
23 } else if (ORG_NAME === 'org2') {
24   CLIENT_CONNECTION_PROFILE_PATH = './profiles/org2-client.yaml';
25 } else {
26   console.log("Exiting : Invalid Organization name");
27   exit(0);
28 }
29
30 //timestamp
31 var base_timestamp = 201601251930
32 //global counter
33 var committedTxs = 0;
34
35 // Variable to hold the client
36 var client = {};
37 // Variable to hold the channel
38 var channel = {}
39
40
41 var initial_timer;
42 var hrstart = [];
43 var hrend = [];
44 var total_time = [];
45
46 main()
47
48 async function main() {
49   client = await setupClient();
50
51   channel = await setupChannel();
52
53   initial_timer = process.hrtime();
54   for (var i = 0; i < NUMBER_OF_TXS; i++) {
55     invokeChaincode();
56   }
57   // var hrend = process.hrtime(hrstart)
58   // console.info('Peer : %s : Execution time (hr): %ds %dms', PEER_NAME, hrend
59   d[0], hrend[1] / 1000000)
60 }
61
62 async function invokeChaincode() {

```

```

63
64     let peerName = channel.getChannelPeer(PEER_NAME)
65
66     var tx_id = client.newTransactionID();
67     let tx_id_string = tx_id.getTransactionID();
68
69
70     var meas_info = (Math.floor(10000000 + Math.random() * 90000000)).toString()
71     var counter = (Math.floor(10000000 + Math.random() * 90000000)).toString()
72     var cell_name = "8424bf520db261335d52a0b827a78538"
73     var value = (Math.floor(Math.random() * 100)).toString()
74
75     var timestamp = base_timestamp++
76     var timestamp_string = timestamp.toString()
77
78     var request = {
79         targets: peerName,
80         chaincodeId: CHAINCODE_ID,
81         fcn: 'write',
82         args: [meas_info, counter, cell_name, value, timestamp_string],
83         chainId: CHANNEL_NAME,
84         txId: tx_id
85     };
86
87     hrstart.push(process.hrtime());
88     // console.log("#1 channel.sendTransactionProposal Done.")
89     let results = await channel.sendTransactionProposal(request);
90
91     // Array of proposal responses
92     var proposalResponses = results[0];
93
94     var proposal = results[1];
95
96     var all_good = true;
97     for (var i in proposalResponses) {
98         let good = false
99         if (proposalResponses[i].response && proposalResponses[i].response.status === 200) {
100             good = true;
101             // console.log(`\tinvoke chaincode EP response #${i} was good`);
102         } else {
103             // console.log(`\tinvoke chaincode EP response #${i} was bad!!!`);
104         }
105         all_good = all_good & good
106     }
107     // console.log("#2 Looped through the EP results all_good=", all_good)
108
109     await setupTxListener(tx_id_string)
110     // console.log('#3 Registered the Tx Listener')
111
112     var orderer_request = {
113         txId: tx_id,
114         proposalResponses: proposalResponses,
115         proposal: proposal
116     };
117
118     await channel.sendTransaction(orderer_request);
119     // console.log("#4 channel.sendTransaction - waiting for Tx Event")
120
121 }
122
123
124
125 function setupTxListener(tx_id_string) {

```

```

126     let event_hub = channel.getChannelEventHub(PEER_NAME);
127
128     event_hub.registerTxEvent(tx_id_string, (tx, code, block_num) => {
129         // console.log("#5 Received Tx Event")
130         // console.log('\tThe chaincode invoke chaincode transaction has been co
mmitted on peer %s', event_hub.getPeerAddr());
131         // console.log('\tTransaction %s is in block %s', tx, block_num);
132
133         if (code !== 'VALID') {
134             // console.log('\tThe invoke chaincode transaction was invalid, code
:%s', code);
135         } else {
136             // console.log('\tThe invoke chaincode transaction was VALID.');
```



```
187 }  
188  
189  
190 async function setupChannel() {  
191   try {  
192     channel = await client.getChannel(CHANNEL_NAME, true)  
193   } catch (e) {  
194     console.log("Could NOT create channel: ", CHANNEL_NAME)  
195     process.exit(1)  
196   }  
197  
198   return channel  
199 }
```

```

1  'use strict';
2  const fs = require('fs');
3  const Client = require('fabric-client');
4  const { exit } = require('process');
5
6  var myArgs = process.argv.slice(2);//get arguments from command line
7
8  // Arguments Parsing
9  const ORG_NAME = myArgs[0];
10 const USER_NAME = myArgs[1];
11 const PEER_NAME = myArgs[2];
12 const CHANNEL_NAME = myArgs[3];
13 const CHAINCODE_ID = myArgs[4];
14 const NUMBER_OF_TXS = parseInt(myArgs[5]);
15
16
17 // Constants for profile
18 const CONNECTION_PROFILE_PATH = './profiles/dev-connect.yaml';
19
20 var CLIENT_CONNECTION_PROFILE_PATH;
21 // Client section configuration
22 if (ORG_NAME === 'org1') {
23     CLIENT_CONNECTION_PROFILE_PATH = './profiles/org1-client.yaml';
24 } else if (ORG_NAME === 'org2') {
25     CLIENT_CONNECTION_PROFILE_PATH = './profiles/org2-client.yaml';
26 } else {
27     console.log("Exiting : Invalid Organization name");
28     exit(0);
29 }
30
31 //global counter
32 var numTxs = 0;
33
34 // Variable to hold the client
35 var client = {};
36 // Variable to hold the channel
37 var channel = {};
38
39 var initial_timer;
40 var hrstart = [];
41 var hrend = [];
42 var total_time = [];
43
44 main()
45
46 async function main() {
47     client = await setupClient();
48
49     channel = await setupChannel();
50
51     let id = "{\"selector\":{\"_id\":{\"$eq\":\"50331648_50331662\"}}}\"";
52
53     initial_timer = process.hrtime();
54     for (var i = 0; i < NUMBER_OF_TXS; i++) {
55         queryImage(id); //async call
56     }
57
58 }
59
60
61
62 async function queryImage(id) {
63

```

Jan 09, 21 13:14	testRAT_query.js	Page 2/3
<pre> 64 let peerName = channel.getChannelPeer(PEER_NAME) 65 66 let request = { 67 targets: peerName, 68 chaincodeId: CHAINCODE_ID, 69 fcn: 'queryRecords', 70 args: [id] 71 }; 72 73 // send the query proposal to the peer 74 hrstart.push(process.hrtime()); 75 let response = await channel.queryByChaincode(request); 76 hrend.push(process.hrtime(hrstart[numTxs])); //diff of start and end time 77 numTxs++; 78 if (numTxs === NUMBER_OF_TXS) { 79 var final_timer = process.hrtime(initial_timer); 80 console.log("\t\t", final_timer); 81 for (var timer of hrend) { 82 total_time.push(timer[0] * 1000 + timer[1] / 1000000); 83 } 84 85 var sum = total_time.reduce((acc, c) => acc + c, 0); 86 var average = sum / NUMBER_OF_TXS; 87 88 //console.log("\t\tAverage Retrieval Latency = %d ms", average); 89 //console.log("\t\tQueried Txs = ", numTxs); 90 exit(0); 91 } 92 93 return 94 } 95 96 async function setupClient() { 97 98 const client = Client.loadFromConfig(CONNECTION_PROFILE_PATH) 99 100 client.loadFromConfig(CLIENT_CONNECTION_PROFILE_PATH) 101 102 await client.initCredentialStores() 103 .then((done) => { 104 // console.log("initCredentialStore(): ", done) 105 }) 106 107 let userContext = await client.loadUserFromStateStore(USER_NAME) 108 if (userContext == null) { 109 console.log("User NOT found in credstore: ", USER_NAME) 110 process.exit(1) 111 } 112 113 client.setUserContext(userContext, true) 114 115 return client 116 } 117 118 async function setupChannel() { 119 try { 120 channel = await client.getChannel(CHANNEL_NAME, true) 121 } catch (e) { 122 console.log("Could NOT create channel: ", CHANNEL_NAME) 123 process.exit(1) 124 } 125 } 126 </pre>		
Jan 09, 21 13:14	testRAT_query.js	Page 3/3
<pre> 127 return channel 128 } </pre>		

```
1  'use strict';
2
3  const { Contract } = require('fabric-contract-api');
4
5
6  class contract_mlModel extends Contract {
7
8      async Init(ctx) {
9
10     }
11
12     async writeModel(ctx, id, model_json, model_weights) {
13
14         const Telco_Model = {
15             ID: id,
16             Model_Json: model_json,
17             Model_Weights: model_weights,
18         };
19
20         return ctx.stub.putState(id, Buffer.from(JSON.stringify(Telco_Model)));
21     }
22
23     // ReadAsset returns the asset stored in the world state with given id.
24     async queryModel(ctx, id) {
25
26         const Telco_Model = await ctx.stub.getState(id);
27         if (!Telco_Model || Telco_Model.length === 0) {
28             throw new Error('The Model ${id} does not exist');
29         }
30
31         var record = JSON.parse(Telco_Model)
32
33         var model_buffer = new Buffer(record.Model_Json, 'base64');
34         var weights_buffer = new Buffer(record.Model_Weights, 'base64');
35
36         return model_buffer.toString();
37     }
38 }
39
40 }
41
42 module.exports = contract_mlModel;
43
44
```

```

1  package main
2
3  import (
4      "encoding/json"
5      "fmt"
6      "strconv"
7      "strings"
8      "time"
9
10     "github.com/hyperledger/fabric-chaincode-go/shim"
11     "github.com/hyperledger/fabric-protos-go/peer"
12 )
13
14 //SmartContract object! usually empty
15 type SmartContract struct {
16 }
17
18 //For this implementaion I assumed that key is
19 //MeasInfo and Counter. We use this struct to
20 //store data in the ledger.
21 type TelcoEntry struct {
22     MeasInfo string 'json:"meas_info"' //col 1
23     Counter string 'json:"counter"' //col 2
24     CellName string 'json:"cell_name"' //col 5
25     Value float32 'json:"value"' //col 6
26     Timestamp int64 'json:"timestamp"' //col 7, e.g. '201512200045' -> '%Y
27     %m%d%H%M', this format can be used with numeric comparisons
28 }
29
30 type RecordHistory struct{
31     TxId string 'json:"tx_id"'
32     Timestamp string 'json:"timestamp"'
33     Record TelcoEntry 'json:"record"'
34 }
35
36 // instantiates smart contract
37 func (s *SmartContract) Init(stub shim.ChaincodeStubInterface) peer.Response {
38     fmt.Println("Chaincode instantiated")
39     return shim.Success(nil)
40 }
41
42 // this functions is the chaincode entry point
43 func (s *SmartContract) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
44     function, args := stub.GetFunctionAndParameters()
45
46     if function == "write" {
47         return s.write(stub, args)
48     } else if function == "queryRecords" {
49         return s.queryRecords(stub, args)
50     } else if function == "queryRecordsWithPagination" {
51         return s.queryRecordsWithPagination(stub, args)
52     } else if function == "getRecordHistory" {
53         return s.getRecordHistory(stub, args)
54     }
55
56     return shim.Success(nil)
57 }
58
59 // args[0] : meas_info
60 // args[1] : counter
61 // args[2] : cell_name
62 // args[3] : value

```

```

63 // args[4] : timestamp
64 func (s *SmartContract) write(stub shim.ChaincodeStubInterface, args []string) p
eer.Response {
65
66     // ==== Input sanitation ====
67     if len(args) != 5 {
68         return shim.Error("Invalid number of arguments.")
69     }
70
71     meas_info := strings.ToLower(args[0])
72
73     counter := strings.ToLower(args[1])
74
75     cell_name := strings.ToLower(args[2])
76     if (len(cell_name) != 32){
77         return shim.Error("Cell name must have 32 characters length")
78     }
79
80     v, err := strconv.ParseFloat(args[3], 32)
81     if err != nil {
82         return shim.Error("4th argument must be a numeric string")
83     }
84     value := float32(v)
85
86     timestamp, err := strconv.ParseInt(args[4], 10, 64)
87     if err != nil {
88         return shim.Error("5th argument must be a numeric string")
89     }
90
91     // ==== Create record key ====
92     record_id := meas_info + "_" + counter
93
94     // ==== Create Telco Entry object and marshal to JSON ====
95     record := TelcoEntry{
96         MeasInfo: meas_info,
97         Counter: counter,
98         CellName: cell_name,
99         Value : value,
100         Timestamp : timestamp,
101     }
102
103     recordAsBytes, err := json.Marshal(record)
104     if err != nil {
105         return shim.Error(err.Error())
106     }
107
108     // === Save record to world state ===
109     err = stub.PutState(record_id, recordAsBytes)
110     if err != nil {
111         return shim.Error(err.Error())
112     }
113
114     return shim.Success(nil)
115 }
116
117 // eg query string {"selector":{"$timestamp" {"$gt" : 0}}
118 // this query retrieves all records with timestamp greater than 0
119 // args[0] : queryString
120
121 // Need debugging
122 // Error: endorsement failure during invoke. response: status:500 message:
123 // "GET QUERY RESULT failed: transaction ID: 9140274f4bd77dc877871f8dfca8eedd929
3ad1694c31b6dbd9cfaafc628d124: invalid character '{' after object key"

```

```

124 func (s *SmartContract) queryRecords(stub shim.ChaincodeStubInterface, args []string) peer.Response {
125
126     fmt.Println("I am in")
127     if len(args) < 1 {
128         return shim.Error("Incorrect number of arguments. Expecting only
129         1")
130     }
131     queryString := args[0]
132
133     resultsIterator, err := stub.GetQueryResult(queryString)
134     if err != nil {
135         return shim.Error(err.Error())
136     }
137     defer resultsIterator.Close()
138
139     var records []TelcoEntry
140
141     records, err = constructQueryResponseFromIterator(resultsIterator, records)
142     if err != nil {
143         return shim.Error(err.Error())
144     }
145
146     recordsBytes, err := json.Marshal(records)
147     if err != nil {
148         return shim.Error("Marshal failed!")
149     }
150
151     fmt.Println(recordsBytes)
152
153     return shim.Success(recordsBytes)
154 }
155
156 // eg query string {"selector":{"$timestamp" {"$gt" : 0}}}
157 // this query retrieves all records with timestamp greater than 0
158 // args[0] : queryString
159 // args[1] : pageSize
160 // args[2] : bookmark
161 func (s *SmartContract) queryRecordsWithPagination(stub shim.ChaincodeStubInterface, args []string) peer.Response {
162     // ==== Input sanitation ====
163     queryString := args[0]
164
165     pageSize, err := strconv.ParseInt(args[1], 10, 32)
166     if err != nil {
167         return shim.Error(err.Error())
168     }
169
170     bookmark := args[2]
171
172     var records []TelcoEntry
173     loop := true
174
175     for loop {
176
177         resultsIterator, responseMetadata, err := stub.GetQueryResultWithPagination(queryString, int32(pageSize), bookmark)
178         if err != nil {
179             return shim.Error(err.Error())
180         }
181         defer resultsIterator.Close()

```

```

182
183     records, err = constructQueryResponseFromIterator(resultsIterato
r, records)
184     if err != nil {
185         return shim.Error(err.Error())
186     }
187
188     if responseMetadata.FetchedRecordsCount == 0 {
189         loop = false
190     }
191
192     bookmark = responseMetadata.Bookmark
193
194     fmt.Println("paginated results")
195 }
196
197 recordsBytes, err := json.Marshal(records)
198 if err != nil {
199     return shim.Error("Marshal failed!")
200 }
201
202 fmt.Println(recordsBytes)
203
204 return shim.Success(recordsBytes)
205 }
206
207 //args[0] : record_id
208 func (s *SmartContract) getRecordHistory(stub shim.ChaincodeStubInterface, args
[]string) peer.Response {
209
210     if len(args) != 1 {
211         return shim.Error("Incorrect number of arguments. Expecting only
1")
212     }
213
214     record_id := args[0]
215
216     resultsIterator, err := stub.GetHistoryForKey(record_id)
217     if err != nil {
218         return shim.Error(err.Error())
219     }
220     defer resultsIterator.Close()
221
222     var historyArray []RecordHistory
223
224     for resultsIterator.HasNext() {
225         response, err := resultsIterator.Next()
226         if err != nil {
227             return shim.Error(err.Error())
228         }
229
230         var recordInstance RecordHistory
231
232         recordInstance.TxId = response.TxId
233         recordInstance.Timestamp = time.Unix(response.Timestamp.Seconds,
int64(response.Timestamp.Nanos)).String()
234
235         var value TelcoEntry
236         err = json.Unmarshal(response.Value, &value)
237         if err != nil {
238             return shim.Error(err.Error())
239         }
240         recordInstance.Record = value

```



```
241         historyArray = append(historyArray, recordInstance)
242     }
243
244     responseBytes, err := json.Marshal(historyArray)
245     if err != nil {
246         return shim.Error("Marshal failed!")
247     }
248
249     fmt.Println(responseBytes)
250
251     return shim.Success(responseBytes)
252 }
253
254 // this is an utility function
255 func constructQueryResponseFromIterator(resultsIterator shim.StateQueryIteratorI
256 nterface, recordsArray []TelcoEntry) ([]TelcoEntry, error) {
257
258     // Get the query results
259     for resultsIterator.HasNext() {
260         queryResponse, err := resultsIterator.Next()
261         if err != nil {
262             return nil, err
263         }
264
265         // Get the retrieved record
266         var record TelcoEntry
267         err = json.Unmarshal(queryResponse.Value, &record)
268         if err != nil {
269             return nil, err
270         }
271
272         recordsArray = append(recordsArray, record)
273     }
274
275     return recordsArray, nil
276 }
277
278 func main() {
279     err := shim.Start(new(SmartContract))
280     if err != nil {
281         fmt.Printf("Error when starting chaincode : %s", err)
282     }
283 }
284 }
```

```

1 Organizations:
2   - &OrdererOrg
3     Name: OrdererOrg
4     # ID to load the MSP definition as
5     ID: OrdererMSP
6     # MSPDir is the filesystem path which contains the MSP configuration
7     MSPDir: ../organizations/ordererOrganizations/example.com/msp
8
9     # Policies defines the set of policies at this level of the config tree
10    # For organization policies, their canonical path is usually
11    # /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
12    Policies:
13      Readers:
14        Type: Signature
15        Rule: "OR('OrdererMSP.member')"
16      Writers:
17        Type: Signature
18        Rule: "OR('OrdererMSP.member')"
19      Admins:
20        Type: Signature
21        Rule: "OR('OrdererMSP.admin')"
22
23    OrdererEndpoints:
24      - orderer.example.com:7050
25
26  - &Org1
27    # DefaultOrg defines the organization which is used in the sampleconfig
28    # of the fabric.git development environment
29    Name: Org1MSP
30    # ID to load the MSP definition as
31    ID: Org1MSP
32    MSPDir: ../organizations/peerOrganizations/org1.example.com/msp
33
34    # Policies defines the set of policies at this level of the config tree
35    # For organization policies, their canonical path is usually
36    # /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
37    Policies:
38      Readers:
39        Type: Signature
40        Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
41      Writers:
42        Type: Signature
43        Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
44      Admins:
45        Type: Signature
46        Rule: "OR('Org1MSP.admin')"
47      Endorsement:
48        Type: Signature
49        Rule: "OR('Org1MSP.peer')"
50
51    # leave this flag set to true.
52    AnchorPeers:
53      # AnchorPeers defines the location of peers which can be used
54      # for cross org gossip communication. Note, this value is only
55      # encoded in the genesis block in the Application section context
56      - Host: peer0.org1.example.com
57        Port: 7051
58
59  - &Org2
60    # DefaultOrg defines the organization which is used in the sampleconfig
61    # of the fabric.git development environment
62    Name: Org2MSP
63    # ID to load the MSP definition as

```

```

64     ID: Org2MSP
65     MSPDir: ../organizations/peerOrganizations/org2.example.com/msp
66
67     # Policies defines the set of policies at this level of the config tree
68     # For organization policies, their canonical path is usually
69     # /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
70     Policies:
71       Readers:
72         Type: Signature
73         Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP.client')"
74       Writers:
75         Type: Signature
76         Rule: "OR('Org2MSP.admin', 'Org2MSP.client')"
77       Admins:
78         Type: Signature
79         Rule: "OR('Org2MSP.admin')"
80       Endorsement:
81         Type: Signature
82         Rule: "OR('Org2MSP.peer')"
83
84     AnchorPeers:
85       # AnchorPeers defines the location of peers which can be used
86       # for cross org gossip communication. Note, this value is only
87       # encoded in the genesis block in the Application section context
88       - Host: peer0.org2.example.com
89         Port: 9051
90
91     #####
92     #
93     # SECTION: Capabilities
94     #
95     # - This section defines the capabilities of fabric network. This is a new
96     # concept as of v1.1.0 and should not be utilized in mixed networks with
97     # v1.0.x peers and orderers. Capabilities define features which must be
98     # present in a fabric binary for that binary to safely participate in the
99     # fabric network. For instance, if a new MSP type is added, newer binaries
100    # might recognize and validate the signatures from this type, while older
101    # binaries without this support would be unable to validate those
102    # transactions. This could lead to different versions of the fabric binaries
103    # having different world states. Instead, defining a capability for a channel
104    # informs those binaries without this capability that they must cease
105    # processing transactions until they have been upgraded. For v1.0.x if any
106    # capabilities are defined (including a map with all capabilities turned off)
107    # then the v1.0.x peer will deliberately crash.
108    #
109    #####
110    Capabilities:
111      # Channel capabilities apply to both the orderers and the peers and must be
112      # supported by both.
113      # Set the value of the capability to true to require it.
114      Channel: &ChannelCapabilities
115        # V2_0 capability ensures that orderers and peers behave according
116        # to v2.0 channel capabilities. Orderers and peers from
117        # prior releases would behave in an incompatible way, and are therefore
118        # not able to participate in channels at v2.0 capability.
119        # Prior to enabling V2.0 channel capabilities, ensure that all
120        # orderers and peers on a channel are at v2.0.0 or later.
121        V2_0: true
122
123      # Orderer capabilities apply only to the orderers, and may be safely
124      # used with prior release peers.
125      # Set the value of the capability to true to require it.
126      Orderer: &OrdererCapabilities

```

```

127     # V2_0 orderer capability ensures that orderers behave according
128     # to v2.0 orderer capabilities. Orderers from
129     # prior releases would behave in an incompatible way, and are therefore
130     # not able to participate in channels at v2.0 orderer capability.
131     # Prior to enabling V2.0 orderer capabilities, ensure that all
132     # orderers on channel are at v2.0.0 or later.
133     V2_0: true
134
135     # Application capabilities apply only to the peer network, and may be safely
136     # used with prior release orderers.
137     # Set the value of the capability to true to require it.
138     Application: &ApplicationCapabilities
139     # V2_0 application capability ensures that peers behave according
140     # to v2.0 application capabilities. Peers from
141     # prior releases would behave in an incompatible way, and are therefore
142     # not able to participate in channels at v2.0 application capability.
143     # Prior to enabling V2.0 application capabilities, ensure that all
144     # peers on channel are at v2.0.0 or later.
145     V2_0: true
146
147
148     #####
149     #
150     # SECTION: Application
151     #
152     # - This section defines the values to encode into a config transaction or
153     # genesis block for application related parameters
154     #
155     #####
156     Application: &ApplicationDefaults
157
158     # Organizations is the list of orgs which are defined as participants on
159     # the application side of the network
160     Organizations:
161
162     # Policies defines the set of policies at this level of the config tree
163     # For Application policies, their canonical path is
164     # /Channel/Application/<PolicyName>
165     Policies:
166         Readers:
167             Type: ImplicitMeta
168             Rule: "ANY Readers"
169         Writers:
170             Type: ImplicitMeta
171             Rule: "ANY Writers"
172         Admins:
173             Type: ImplicitMeta
174             Rule: "MAJORITY Admins"
175         LifecycleEndorsement:
176             Type: ImplicitMeta
177             Rule: "ANY Endorsement"
178         Endorsement:
179             Type: ImplicitMeta
180             Rule: "ANY Endorsement"
181
182     Capabilities:
183         <<: *ApplicationCapabilities
184     #####
185     #
186     # SECTION: Orderer
187     #
188     # - This section defines the values to encode into a config transaction or
189     # genesis block for orderer related parameters

```

```

190 #
191 #####
192 Orderer: &OrdererDefaults
193
194 # Orderer Type: The orderer implementation to start
195 OrdererType: etcdraft
196
197 Addresses:
198   - orderer.example.com:7050
199
200 EtcdRaft:
201   Consenters:
202     - Host: orderer.example.com
203       Port: 7050
204       ClientTLS Cert: ../organizations/ordererOrganizations/example.com/order
205 ers/orderer.example.com/tls/server.crt
206       ServerTLS Cert: ../organizations/ordererOrganizations/example.com/order
207 ers/orderer.example.com/tls/server.crt
208
209 # Batch Timeout: The amount of time to wait before creating a batch
210 BatchTimeout: 2s
211
212 # Batch Size: Controls the number of messages batched into a block
213 BatchSize:
214   # Max Message Count: The maximum number of messages to permit in a batch
215   MaxMessageCount: 10
216   # Absolute Max Bytes: The absolute maximum number of bytes allowed for
217   # the serialized messages in a batch.
218   AbsoluteMaxBytes: 99 MB
219   # Preferred Max Bytes: The preferred maximum number of bytes allowed for
220   # the serialized messages in a batch. A message larger than the preferre
221 d
222   # max bytes will result in a batch larger than preferred max bytes.
223   PreferredMaxBytes: 512 KB
224
225 # Organizations is the list of orgs which are defined as participants on
226 # the orderer side of the network
227 Organizations:
228
229 # Policies defines the set of policies at this level of the config tree
230 # For Orderer policies, their canonical path is
231 # /Channel/Orderer/<PolicyName>
232 Policies:
233   Readers:
234     Type: ImplicitMeta
235     Rule: "ANY Readers"
236   Writers:
237     Type: ImplicitMeta
238     Rule: "ANY Writers"
239   Admins:
240     Type: ImplicitMeta
241     Rule: "MAJORITY Admins"
242   # BlockValidation specifies what signatures must be included in the bloc
243 k
244   # from the orderer for the peer to validate it.
245   BlockValidation:
246     Type: ImplicitMeta
247     Rule: "ANY Writers"
248 #####

```

```

249 #
250 # CHANNEL
251 #
252 # This section defines the values to encode into a config transaction or
253 # genesis block for channel related parameters.
254 #
255 #####
256 Channel: &ChannelDefaults
257 # Policies defines the set of policies at this level of the config tree
258 # For Channel policies, their canonical path is
259 # /Channel/<PolicyName>
260 Policies:
261 # Who may invoke the 'Deliver' API
262 Readers:
263     Type: ImplicitMeta
264     Rule: "ANY Readers"
265 # Who may invoke the 'Broadcast' API
266 Writers:
267     Type: ImplicitMeta
268     Rule: "ANY Writers"
269 # By default, who may modify elements at this config level
270 Admins:
271     Type: ImplicitMeta
272     Rule: "MAJORITY Admins"
273
274 # Capabilities describes the channel level capabilities, see the
275 # dedicated Capabilities section elsewhere in this file for a full
276 # description
277 Capabilities:
278     <<: *ChannelCapabilities
279
280 #####
281 #
282 # Profile
283 #
284 # - Different configuration profiles may be encoded here to be specified
285 # as parameters to the configtxgen tool
286 #
287 #####
288 Profiles:
289
290     TwoOrgsOrdererGenesis:
291         <<: *ChannelDefaults
292         Orderer:
293             <<: *OrdererDefaults
294             Organizations:
295                 - *OrdererOrg
296             Capabilities:
297                 <<: *OrdererCapabilities
298         Consortiums:
299             SampleConsortium:
300                 Organizations:
301                     - *Org1
302                     - *Org2
303
304     BasicChannel:
305         Consortium: SampleConsortium
306         <<: *ChannelDefaults
307         Application:
308             <<: *ApplicationDefaults
309             Organizations:
310                 - *Org1
311                 - *Org2
312             Capabilities:

```

```

312         <<: *ApplicationCapabilities

```

```

1 #####
2 #
3 #   Peer section
4 #
5 #####
6 peer:
7
8     # The peer id provides a name for this peer instance and is used when
9     # naming docker resources.
10    id: jdoe
11
12    # The networkId allows for logical separation of networks and is used when
13    # naming docker resources.
14    networkId: dev
15
16    # The Address at local network interface this Peer will listen on.
17    # By default, it will listen on all network interfaces
18    listenAddress: 0.0.0.0:7051
19
20    # The endpoint this peer uses to listen for inbound chaincode connections.
21    # If this is commented-out, the listen address is selected to be
22    # the peer's address (see below) with port 7052
23    # chaincodeListenAddress: 0.0.0.0:7052
24
25    # The endpoint the chaincode for this peer uses to connect to the peer.
26    # If this is not specified, the chaincodeListenAddress address is selected.
27    # And if chaincodeListenAddress is not specified, address is selected from
28    # peer listenAddress.
29    # chaincodeAddress: 0.0.0.0:7052
30
31    # When used as peer config, this represents the endpoint to other peers
32    # in the same organization. For peers in other organization, see
33    # gossip.externalEndpoint for more info.
34    # When used as CLI config, this means the peer's endpoint to interact with
35    address: 0.0.0.0:7051
36
37    # Whether the Peer should programmatically determine its address
38    # This case is useful for docker containers.
39    addressAutoDetect: false
40
41    # Keepalive settings for peer server and clients
42    keepalive:
43        # Interval is the duration after which if the server does not see
44        # any activity from the client it pings the client to see if it's alive
45        interval: 7200s
46        # Timeout is the duration the server waits for a response
47        # from the client after sending a ping before closing the connection
48        timeout: 20s
49        # MinInterval is the minimum permitted time between client pings.
50        # If clients send pings more frequently, the peer server will
51        # disconnect them
52        minInterval: 60s
53        # Client keepalive settings for communicating with other peer nodes
54        client:
55            # Interval is the time between pings to peer nodes. This must
56            # be greater than or equal to the minInterval specified by peer
57            # nodes
58            interval: 60s
59            # Timeout is the duration the client waits for a response from
60            # peer nodes before closing the connection
61            timeout: 20s
62        # DeliveryClient keepalive settings for communication with ordering
63        # nodes.

```

```

64     deliveryClient:
65         # Interval is the time between pings to ordering nodes. This must
66         # greater than or equal to the minInterval specified by ordering
67         # nodes.
68         interval: 60s
69         # Timeout is the duration the client waits for a response from
70         # ordering nodes before closing the connection
71         timeout: 20s
72
73
74     # Gossip related configuration
75     gossip:
76         # Bootstrap set to initialize gossip with.
77         # This is a list of other peers that this peer reaches out to at startup
78         # Important: The endpoints here have to be endpoints of peers in the sam
79         # organization, because the peer would refuse connecting to these endpoi
80         # unless they are in the same organization as the peer.
81         bootstrap: 127.0.0.1:7051
82
83         # NOTE: orgLeader and useLeaderElection parameters are mutual exclusive.
84         # Setting both to true would result in the termination of the peer
85         # since this is undefined state. If the peers are configured with
86         # useLeaderElection=false, make sure there is at least 1 peer in the
87         # organization that its orgLeader is set to true.
88
89         # Defines whenever peer will initialize dynamic algorithm for
90         # "leader" selection, where leader is the peer to establish
91         # connection with ordering service and use delivery protocol
92         # to pull ledger blocks from ordering service. It is recommended to
93         # use leader election for large networks of peers.
94         useLeaderElection: true
95         # Statically defines peer to be an organization "leader",
96         # where this means that current peer will maintain connection
97         # with ordering service and disseminate block across peers in
98         # its own organization
99         orgLeader: false
100
101         # Interval for membershipTracker polling
102         membershipTrackerInterval: 5s
103
104         # Overrides the endpoint that the peer publishes to peers
105         # in its organization. For peers in foreign organizations
106         # see 'externalEndpoint'
107         endpoint:
108         # Maximum count of blocks stored in memory
109         maxBlockCountToStore: 100
110         # Max time between consecutive message pushes(unit: millisecond)
111         maxPropagationBurstLatency: 10ms
112         # Max number of messages stored until a push is triggered to remote peer
113         maxPropagationBurstSize: 10
114         # Number of times a message is pushed to remote peers
115         propagateIterations: 1
116         # Number of peers selected to push messages to
117         propagatePeerNum: 3
118         # Determines frequency of pull phases(unit: second)
119         # Must be greater than digestWaitTime + responseWaitTime
120         pullInterval: 4s
121         # Number of peers to pull from
122         pullPeerNum: 3

```


Jan 09, 21 13:18	core.yaml	Page 3/12
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173	<pre> # Determines frequency of pulling state info messages from peers(unit: s econd) requestStateInfoInterval: 4s # Determines frequency of pushing state info messages to peers(unit: sec ond) publishStateInfoInterval: 4s # Maximum time a stateInfo message is kept until expired stateInfoRetentionInterval: # Time from startup certificates are included in Alive messages(unit: se cond) publishCertPeriod: 10s # Should we skip verifying block messages or not (currently not in use) skipBlockVerification: false # Dial timeout(unit: second) dialTimeout: 3s # Connection timeout(unit: second) connTimeout: 2s # Buffer size of received messages recvBuffSize: 20 # Buffer size of sending messages sendBuffSize: 200 # Time to wait before pull engine processes incoming digests (unit: seco nd) # Should be slightly smaller than requestWaitTime digestWaitTime: 1s # Time to wait before pull engine removes incoming nonce (unit: millisec onds) # Should be slightly bigger than digestWaitTime requestWaitTime: 1500ms # Time to wait before pull engine ends pull (unit: second) responseWaitTime: 2s # Alive check interval(unit: second) aliveTimeInterval: 5s # Alive expiration timeout(unit: second) aliveExpirationTimeout: 25s # Reconnect interval(unit: second) reconnectInterval: 25s # This is an endpoint that is published to peers outside of the organiza tion. # If this isn't set, the peer will not be known to other organizations. externalEndpoint: # Leader election service configuration election: # Longest time peer waits for stable membership during leader electi on startup (unit: second) startupGracePeriod: 15s # Interval gossip membership samples to check its stability (unit: s econd) membershipSampleInterval: 1s # Time passes since last declaration message before peer decides to perform leader election (unit: second) leaderAliveThreshold: 10s # Time between peer sends propose message and declares itself as a l eader (sends declaration message) (unit: second) leaderElectionDuration: 5s pvtData: # pullRetryThreshold determines the maximum duration of time private data corresponding for a given block # would be attempted to be pulled from peers until the block would b e committed without the private data pullRetryThreshold: 60s # As private data enters the transient store, it is associated with </pre>	

```

the peer's ledger's height at that time.
174     # transientstoreMaxBlockRetention defines the maximum difference bet
ween the current ledger's height upon commit,
175     # and the private data residing inside the transient store that is g
uaranteed not to be purged.
176     # Private data is purged from the transient store when blocks with s
equences that are multiples
177     # of transientstoreMaxBlockRetention are committed.
178     transientstoreMaxBlockRetention: 1000
179     # pushAckTimeout is the maximum time to wait for an acknowledgement
from each peer
180     # at private data push at endorsement time.
181     pushAckTimeout: 3s
182     # Block to live pulling margin, used as a buffer
183     # to prevent peer from trying to pull private data
184     # from peers that is soon to be purged in next N blocks.
185     # This helps a newly joined peer catch up to current
186     # blockchain height quicker.
187     btlPullMargin: 10
188     # the process of reconciliation is done in an endless loop, while in
each iteration reconciler tries to
189     # pull from the other peers the most recent missing blocks with a ma
ximum batch size limitation.
190     # reconcileBatchSize determines the maximum batch size of missing pr
ivate data that will be reconciled in a
191     # single iteration.
192     reconcileBatchSize: 10
193     # reconcileSleepInterval determines the time reconciler sleeps from
end of an iteration until the beginning
194     # of the next reconciliation iteration.
195     reconcileSleepInterval: 1m
196     # reconciliationEnabled is a flag that indicates whether private dat
a reconciliation is enable or not.
197     reconciliationEnabled: true
198     # skipPullingInvalidTransactionsDuringCommit is a flag that indicate
s whether pulling of invalid
199     # transaction's private data from other peers need to be skipped dur
ing the commit time and pulled
200     # only through reconciler.
201     skipPullingInvalidTransactionsDuringCommit: false
202
203     # Gossip state transfer related configuration
204     state:
205     # indicates whenever state transfer is enabled or not
206     # default value is true, i.e. state transfer is active
207     # and takes care to sync up missing blocks allowing
208     # lagging peer to catch up to speed with rest network
209     enabled: true
210     # checkInterval interval to check whether peer is lagging behind eno
ugh to
211     # request blocks via state transfer from another peer.
212     checkInterval: 10s
213     # responseTimeout amount of time to wait for state transfer response
from
214     # other peers
215     responseTimeout: 3s
216     # batchSize the number of blocks to request via state transfer from
another peer
217     batchSize: 10
218     # blockBufferSize reflects the size of the re-ordering buffer
219     # which captures blocks and takes care to deliver them in order
220     # down to the ledger layer. The actually buffer size is bounded betw
een

```

```

221         # 0 and 2*blockBufferSize, each channel maintains its own buffer
222         blockBufferSize: 100
223         # maxRetries maximum number of re-tries to ask
224         # for single state transfer request
225         maxRetries: 3
226
227     # TLS Settings
228     tls:
229         # Require server-side TLS
230         enabled: false
231         # Require client certificates / mutual TLS.
232         # Note that clients that are not configured to use a certificate will
233         # fail to connect to the peer.
234         clientAuthRequired: false
235         # X.509 certificate used for TLS server
236         cert:
237             file: tls/server.crt
238         # Private key used for TLS server (and client if clientAuthEnabled
239         # is set to true
240         key:
241             file: tls/server.key
242         # Trusted root certificate chain for tls.cert
243         rootcert:
244             file: tls/ca.crt
245         # Set of root certificate authorities used to verify client certificates
246         clientRootCAs:
247             files:
248                 - tls/ca.crt
249         # Private key used for TLS when making client connections. If
250         # not set, peer.tls.key.file will be used instead
251         clientKey:
252             file:
253         # X.509 certificate used for TLS when making client connections.
254         # If not set, peer.tls.cert.file will be used instead
255         clientCert:
256             file:
257
258     # Authentication contains configuration parameters related to authenticating
259     # client messages
260     authentication:
261         # the acceptable difference between the current server time and the
262         # client's time as specified in a client request message
263         timewindow: 15m
264
265     # Path on the file system where peer will store data (eg ledger). This
266     # location must be access control protected to prevent unintended
267     # modification that might corrupt the peer operations.
268     filePath: /var/hyperledger/production
269
270     # BCCSP (Blockchain crypto provider): Select which crypto implementation or
271     # library to use
272     BCCSP:
273         Default: SW
274         # Settings for the SW crypto provider (i.e. when DEFAULT: SW)
275         SW:
276             # TODO: The default Hash and Security level needs refactoring to be
277             # fully configurable. Changing these defaults requires coordination
278             # SHA2 is hardcoded in several places, not only BCCSP
279             Hash: SHA2
280             Security: 256
281             # Location of Key Store
282             FileKeyStore:
283                 # If "", defaults to 'mspConfigPath'/keystore

```

```

284         KeyStore:
285         # Settings for the PKCS#11 crypto provider (i.e. when DEFAULT: PKCS11)
286         PKCS11:
287             # Location of the PKCS11 module library
288             Library:
289             # Token Label
290             Label:
291             # User PIN
292             Pin:
293             Hash:
294             Security:
295
296         # Path on the file system where peer will find MSP local configurations
297         mspConfigPath: msp
298
299         # Identifier of the local MSP
300         # ----!!!IMPORTANT!!!-!!!IMPORTANT!!!-!!!IMPORTANT!!!!----
301         # Deployers need to change the value of the localMspId string.
302         # In particular, the name of the local MSP ID of a peer needs
303         # to match the name of one of the MSPs in each of the channel
304         # that this peer is a member of. Otherwise this peer's messages
305         # will not be identified as valid by other nodes.
306         localMspId: SampleOrg
307
308         # CLI common client config options
309         client:
310             # connection timeout
311             connTimeout: 3s
312
313         # Delivery service related config
314         deliveryclient:
315             # It sets the total time the delivery service may spend in reconnection
316             # attempts until its retry logic gives up and returns an error
317             reconnectTotalTimeThreshold: 3600s
318
319         # It sets the delivery service <-> ordering service node connection time
320         out
321         connTimeout: 3s
322
323         # It sets the delivery service maximal delay between consecutive retries
324         reconnectBackoffThreshold: 3600s
325
326         # A list of orderer endpoint addresses which should be overridden
327         # when found in channel configurations.
328         addressOverrides:
329             # - from:
330             #   to:
331             #   caCertsFile:
332             # - from:
333             #   to:
334             #   caCertsFile:
335
336         # Type for the local MSP - by default it's of type bccsp
337         localMspType: bccsp
338
339         # Used with Go profiling tools only in none production environment. In
340         # production, it should be disabled (eg enabled: false)
341         profile:
342             enabled: false
343             listenAddress: 0.0.0.0:6060
344
345         # Handlers defines custom handlers that can filter and mutate
346         # objects passing within the peer, such as:

```

```

346 # Auth filter - reject or forward proposals from clients
347 # Decorators - append or mutate the chaincode input passed to the chaincode
348 # Endorsers - Custom signing over proposal response payload and its mutation
349 # Valid handler definition contains:
350 # - A name which is a factory method name defined in
351 #   core/handlers/library/library.go for statically compiled handlers
352 # - library path to shared object binary for pluggable filters
353 # Auth filters and decorators are chained and executed in the order that
354 # they are defined. For example:
355 # authFilters:
356 # -
357 #   name: FilterOne
358 #   library: /opt/lib/filter.so
359 # -
360 #   name: FilterTwo
361 # decorators:
362 # -
363 #   name: DecoratorOne
364 # -
365 #   name: DecoratorTwo
366 #   library: /opt/lib/decorator.so
367 # Endorsers are configured as a map that its keys are the endorsement system
   chaincodes that are being overridden.
368 # Below is an example that overrides the default ESCC and uses an endorsement
   t plugin that has the same functionality
369 # as the default ESCC.
370 # If the 'library' property is missing, the name is used as the constructor
   method in the builtin library similar
371 # to auth filters and decorators.
372 # endorsers:
373 #   escc:
374 #     name: DefaultESCC
375 #     library: /etc/hyperledger/fabric/plugin/escc.so
376 handlers:
377   authFilters:
378   -
379     name: DefaultAuth
380   -
381     name: ExpirationCheck # This filter checks identity x509 certificate expiration
382   decorators:
383   -
384     name: DefaultDecorator
385   endorsers:
386     escc:
387       name: DefaultEndorsement
388       library:
389     validators:
390       vscc:
391         name: DefaultValidation
392         library:
393
394 #   library: /etc/hyperledger/fabric/plugin/escc.so
395 # Number of goroutines that will execute transaction validation in parallel.
396 # By default, the peer chooses the number of CPUs on the machine. Set this
397 # variable to override that choice.
398 # NOTE: overriding this value might negatively influence the performance of
399 # the peer so please change this value only if you know what you're doing
400 validatorPoolSize:
401
402 # The discovery service is used by clients to query information about peers,

```

Jan 09, 21 13:18	core.yaml	Page 8/12
403	# such as - which peers have joined a certain channel, what is the latest	
404	# channel config, and most importantly - given a chaincode and a channel,	
405	# what possible sets of peers satisfy the endorsement policy.	
406	discovery:	
407	enabled: true	
408	# Whether the authentication cache is enabled or not.	
409	authCacheEnabled: true	
410	# The maximum size of the cache, after which a purge takes place	
411	authCacheMaxSize: 1000	
412	# The proportion (0 to 1) of entries that remain in the cache after the	
413	cache is purged due to overpopulation	
414	authCachePurgeRetentionRatio: 0.75	
415	# Whether to allow non-admins to perform non channel scoped queries.	
416	# When this is false, it means that only peer admins can perform non cha	
417	nnel scoped queries.	
418	orgMembersAllowedAccess: false	
419	# Limits is used to configure some internal resource limits.	
420	limits:	
421	# Concurrency limits the number of concurrently running system chaincode r	
422	equests.	
423	# This option is only supported for qsc at this time.	
424	concurrency:	
425	qsc: 5000	
426	#####	
427	# VM section	
428	#####	
429	vm:	
430	ing	
431	# Endpoint of the vm management system. For docker can be one of the follow	
432	ing in general	
433	# unix:///var/run/docker.sock	
434	# http://localhost:2375	
435	# https://localhost:2376	
436	endpoint: unix:///var/run/docker.sock	
437	# settings for docker vms	
438	docker:	
439	tls:	
440	enabled: false	
441	ca:	
442	file: docker/ca.crt	
443	cert:	
444	file: docker/tls.crt	
445	key:	
446	file: docker/tls.key	
447		
448		
449	# Enables/disables the standard out/err from chaincode containers for	
450	# debugging purposes	
451	attachStdout: false	
452		
453	# Parameters on creating docker container.	
454	# Container may be efficiently created using ipam & dns-server for clust	
455	er	
456	# NetworkMode - sets the networking mode for the container. Supported	
457	# standard values are: 'host'(default), 'bridge', 'ipvlan', 'none'.	
458	# Dns - a list of DNS servers for the container to use.	
459	# Note: 'Privileged' 'Binds' 'Links' and 'PortBindings' properties of	
460	# Docker Host Config are not supported and will not be used if set.	
	# LogConfig - sets the logging driver (Type) and related options	

```

461     # (Config) for Docker. For more info,
462     # https://docs.docker.com/engine/admin/logging/overview/
463     # Note: Set LogConfig using Environment Variables is not supported.
464     hostConfig:
465         NetworkMode: host
466         Dns:
467             # - 192.168.0.1
468         LogConfig:
469             Type: json-file
470             Config:
471                 max-size: "50m"
472                 max-file: "5"
473         Memory: 2147483648
474
475
476 #####
477 #
478 # Chaincode section
479 #
480 #####
481 chaincode:
482
483     # The id is used by the Chaincode stub to register the executing Chaincode
484     # ID with the Peer and is generally supplied through ENV variables
485     # the 'path' form of ID is provided when installing the chaincode.
486     # The 'name' is used for all other requests and can be any string.
487     id:
488         path:
489         name:
490
491
492     # Generic builder environment, suitable for most chaincode types
493     builder: $(DOCKER_NS)/fabric-ccenv:$(PROJECT_VERSION)
494
495     # Enables/disables force pulling of the base docker images (listed below)
496     # during user chaincode instantiation.
497     # Useful when using moving image tags (such as :latest)
498     pull: false
499
500     go:
501         # go will never need more than baseos
502         runtime: $(DOCKER_NS)/fabric-baseos:$(PROJECT_VERSION)
503
504         # whether or not go chaincode should be linked dynamically
505         dynamicLink: false
506
507     java:
508         # This is an image based on java:openjdk-8 with addition compiler
509         # tools added for java shim layer packaging.
510         # This image is packed with shim layer libraries that are necessary
511         # for Java chaincode runtime.
512         runtime: $(DOCKER_NS)/fabric-javaenv:latest
513
514     node:
515         # This is an image based on node:$(NODE_VER)-alpine
516         runtime: $(DOCKER_NS)/fabric-nodeenv:latest
517
518     # List of directories to treat as external builders and launchers for
519     # chaincode. The external builder detection processing will iterate over the
520     # builders in the order specified below.
521     externalBuilders: []
522
523     # The maximum duration to wait for the chaincode build and install process

```

```

524     # to complete.
525     installTimeout: 300s
526
527     # Timeout duration for starting up a container and waiting for Register
528     # to come through.
529     startuptimeout: 300s
530
531     # Timeout duration for Invoke and Init calls to prevent runaway.
532     # This timeout is used by all chaincodes in all the channels, including
533     # system chaincodes.
534     # Note that during Invoke, if the image is not available (e.g. being
535     # cleaned up when in development environment), the peer will automatically
536     # build the image, which might take more time. In production environment,
537     # the chaincode image is unlikely to be deleted, so the timeout could be
538     # reduced accordingly.
539     executetimeout: 30s
540
541     # There are 2 modes: "dev" and "net".
542     # In dev mode, user runs the chaincode after starting peer from
543     # command line on local machine.
544     # In net mode, peer will run chaincode in a docker container.
545     mode: net
546
547     # keepalive in seconds. In situations where the communication goes through a
548     # proxy that does not support keep-alive, this parameter will maintain conne
549 ction
550     # between peer and chaincode.
551     # A value <= 0 turns keepalive off
552     keepalive: 0
553
554     # system chaincodes whitelist. To add system chaincode "myscc" to the
555     # whitelist, add "myscc: enable" to the list below, and register in
556     # chaincode/importsccs.go
557     system:
558       _lifecycle: enable
559       cscc: enable
560       lscc: enable
561       escc: enable
562       vscc: enable
563       qscc: enable
564
565     # Logging section for the chaincode container
566     logging:
567       # Default level for all loggers within the chaincode container
568       level: info
569       # Override default level for the 'shim' logger
570       shim: warning
571       # Format for the chaincode container logs
572       format: '%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}] %{shortfu
573 nc} -> %{level:.4s} %{id:03x}%{color:reset} %{message}'
574
575     #####
576     #
577     # Ledger section - ledger configuration encompasses both the blockchain
578     # and the state
579     #
580     #####
581     ledger:
582       blockchain:
583       state:
584         # stateDatabase - options are "goleveldb", "CouchDB"

```



```

585 # goleveldb - default state database stored in goleveldb.
586 # CouchDB - store state database in CouchDB
587 stateDatabase: goleveldb
588 # Limit on the number of records to return per query
589 totalQueryLimit: 100000
590 couchDBConfig:
591   # It is recommended to run CouchDB on the same server as the peer, and
592   # not map the CouchDB container port to a server port in docker-compose.
593   # Otherwise proper security must be provided on the connection between
594   # CouchDB client (on the peer) and server.
595   couchDBAddress: 127.0.0.1:5984
596   # This username must have read and write authority on CouchDB
597   username:
598   # The password is recommended to pass as an environment variable
599   # during start up (eg CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD).
600   # If it is stored here, the file must be access control protected
601   # to prevent unintended users from discovering the password.
602   password:
603   # Number of retries for CouchDB errors
604   maxRetries: 3
605   # Number of retries for CouchDB errors during peer startup
606   maxRetriesOnStartup: 12
607   # CouchDB request timeout (unit: duration, e.g. 20s)
608   requestTimeout: 35s
609   # Limit on the number of records per each CouchDB query
610   # Note that chaincode queries are only bound by totalQueryLimit.
611   # Internally the chaincode may execute multiple CouchDB queries,
612   # each of size internalQueryLimit.
613   internalQueryLimit: 1000
614   # Limit on the number of records per CouchDB bulk update batch
615   maxBatchUpdateSize: 1000
616   # Warm indexes after every N blocks.
617   # This option warms any indexes that have been
618   # deployed to CouchDB after every N blocks.
619   # A value of 1 will warm indexes after every block commit,
620   # to ensure fast selector queries.
621   # Increasing the value may improve write efficiency of peer and CouchDB,
622   # but may degrade query response time.
623   warmIndexesAfterNBlocks: 1
624   # Create the _global_changes system database
625   # This is optional. Creating the global changes database will require
626   # additional system resources to track changes and maintain the database
627   createGlobalChangesDB: false
628   # CacheSize denotes the maximum mega bytes (MB) to be allocated for the i
n-memory state
629   # cache. Note that CacheSize needs to be a multiple of 32 MB. If it is no
t a multiple
630   # of 32 MB, the peer would round the size to the next multiple of 32 MB.
631   # To disable the cache, 0 MB needs to be assigned to the cacheSize.
632   cacheSize: 64
633
634   history:
635     # enableHistoryDatabase - options are true or false
636     # Indicates if the history of key updates should be stored.
637     # All history 'index' will be stored in goleveldb, regardless if using
638     # CouchDB or alternate database for the state.
639     enableHistoryDatabase: true
640
641   pvtdataStore:
642     # the maximum db batch size for converting
643     # the ineligible missing data entries to eligible missing data entries
644     collElgProcMaxDbBatchSize: 5000
645     # the minimum duration (in milliseconds) between writing

```

```

646     # two consecutive db batches for converting the ineligible missing data entr
647     ies to eligible missing data entries
648     collElgProcDbBatchesInterval: 1000
649 #####
650 #
651 #     Operations section
652 #
653 #####
654 operations:
655     # host and port for the operations server
656     listenAddress: 127.0.0.1:9443
657
658     # TLS configuration for the operations endpoint
659     tls:
660         # TLS enabled
661         enabled: false
662
663         # path to PEM encoded server certificate for the operations server
664         cert:
665             file:
666
667         # path to PEM encoded server key for the operations server
668         key:
669             file:
670
671         # most operations service endpoints require client authentication when T
672         ion
673         # is enabled. clientAuthRequired requires client certificate authenticat
674         # at the TLS layer to access all resources.
675         clientAuthRequired: false
676         # paths to PEM encoded ca certificates to trust for client authenticatio
677         n
678         clientRootCAs:
679             files: []
680 #####
681 #
682 #     Metrics section
683 #
684 #####
685 metrics:
686     # metrics provider is one of statsd, prometheus, or disabled
687     provider: disabled
688
689     # statsd configuration
690     statsd:
691         # network type: tcp or udp
692         network: udp
693
694         # statsd server address
695         address: 127.0.0.1:8125
696
697         # the interval at which locally cached counters and gauges are pushed
698         # to statsd; timings are pushed immediately
699         writeInterval: 10s
700
701         # prefix is prepended to all emitted statsd metrics
702         prefix:

```

```
1 OrdererOrgs:
2   - Name: Orderer
3     Domain: example.com
4     EnableNodeOUs: true
5     Specs:
6       - Hostname: orderer
7         SANS:
8           - localhost
9
10  PeerOrgs:
11    - Name: Org1
12      Domain: org1.example.com
13      EnableNodeOUs: true
14      Template:
15        Count: 2
16        SANS:
17          - localhost
18      Users:
19        Count: 1
20
21    - Name: Org2
22      Domain : org2.example.com
23      EnableNodeOUs : true
24      Template:
25        Count: 2
26        SANS:
27          - localhost
28      Users:
29        Count: 1
```

```

1  version: '2'
2
3  networks:
4    test:
5
6  services:
7
8
9    couchdb0:
10     container_name: couchdb0
11     image: hyperledger/fabric-couchdb:$IMAGE_TAG
12     # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and pa
13     ssword
14     # for CouchDB. This will prevent CouchDB from operating in an "Admin Party"
15     mode.
16     environment:
17       - COUCHDB_USER=admin
18       - COUCHDB_PASSWORD=adminpw
19     # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB
20     service,
21     # for example map it to utilize Fauxton User Interface in dev environments.
22     ports:
23       - "5984:5984"
24     networks:
25       - test
26
27    peer0.org1.example.com:
28     environment:
29       - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
30       - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
31       - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
32       - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
33     depends_on:
34       - couchdb0
35
36    couchdb1:
37     container_name: couchdb1
38     image: hyperledger/fabric-couchdb:$IMAGE_TAG
39     # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and pa
40     ssword
41     # for CouchDB. This will prevent CouchDB from operating in an "Admin Party"
42     mode.
43     environment:
44       - COUCHDB_USER=admin
45       - COUCHDB_PASSWORD=adminpw
46     # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB
47     service,
48     # for example map it to utilize Fauxton User Interface in dev environments.
49     ports:
50       - "6984:5984"
51     networks:
52       - test
53
54    peer1.org1.example.com:
55     environment:
56       - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
57       - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
58       - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
59       - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
60     depends_on:
61       - couchdb1
62
63    couchdb2:

```

```
58     container_name: couchdb2
59     image: hyperledger/fabric-couchdb:$IMAGE_TAG
60     # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and pa
ssword
61     # for CouchDB. This will prevent CouchDB from operating in an "Admin Party"
mode.
62     environment:
63         - COUCHDB_USER=admin
64         - COUCHDB_PASSWORD=adminpw
65     # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB
service,
66     # for example map it to utilize Fauxton User Interface in dev environments.
67     ports:
68         - "7984:5984"
69     networks:
70         - test
71
72 peer0.org2.example.com:
73     environment:
74         - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
75         - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
76         - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
77         - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
78     depends_on:
79         - couchdb2
80
81 couchdb3:
82     container_name: couchdb3
83     image: hyperledger/fabric-couchdb:$IMAGE_TAG
84     # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and pa
ssword
85     # for CouchDB. This will prevent CouchDB from operating in an "Admin Party"
mode.
86     environment:
87         - COUCHDB_USER=admin
88         - COUCHDB_PASSWORD=adminpw
89     # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB
service,
90     # for example map it to utilize Fauxton User Interface in dev environments.
91     ports:
92         - "8984:5984"
93     networks:
94         - test
95
96 peer1.org2.example.com:
97     environment:
98         - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
99         - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb3:5984
100         - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
101         - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
102     depends_on:
103         - couchdb3
```

```

1  version: '2'
2
3  volumes:
4    orderer.example.com:
5    peer0.org1.example.com:
6    peer1.org1.example.com:
7    peer0.org2.example.com:
8    peer1.org2.example.com:
9
10 networks:
11   test:
12
13 services:
14
15   grpc_server:
16     container_name: grpc_server
17     image: python:latest
18     ports:
19       - "9999:9999"
20     command: bash -c "python -m pip install --upgrade pip && python -m pip install grpcio && python -m pip install grpcio-tools && python /root/grpc_server/py/server.py "
21     volumes:
22       - ../../app/gRPC:/root/grpc_server
23     networks:
24       - test
25
26   orderer.example.com:
27     container_name: orderer.example.com
28     image: hyperledger/fabric-orderer:$IMAGE_TAG
29     environment:
30       - FABRIC_LOGGING_SPEC=INFO
31       - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
32       - ORDERER_GENERAL_LISTENPORT=7050
33       - ORDERER_GENERAL_GENESISMETHOD=file
34       - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
35   ck
36     - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
37     - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
38     # enabled TLS
39     - ORDERER_GENERAL_TLS_ENABLED=true
40     - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
41     - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
42     - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
43     - ORDERER_KAFKA_TOPIC_REPLICATIONFACTOR=1
44     - ORDERER_KAFKA_VERBOSE=true
45     - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/server.crt
46     - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/server.key
47     - ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
48     working_dir: /opt/gopath/src/github.com/hyperledger/fabric
49     command: orderer
50     volumes:
51       - ../system-genesis-block/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
52       - ../organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp
53       - ../organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls:/var/hyperledger/orderer/tls
54       - orderer.example.com:/var/hyperledger/production/orderer
55     ports:

```

```

56     - 7050:7050
57     networks:
58     - test
59
60     peer0.org1.example.com:
61     container_name: peer0.org1.example.com
62     image: hyperledger/fabric-peer:$IMAGE_TAG
63     environment:
64     #Generic peer variables
65     - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
66     # the following setting starts chaincode containers on the same
67     # bridge network as the peers
68     # https://docs.docker.com/compose/networking/
69     - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_test
70     - FABRIC_LOGGING_SPEC=INFO
71     #- FABRIC_LOGGING_SPEC=DEBUG
72     - CORE_PEER_TLS_ENABLED=true
73     - CORE_PEER_PROFILE_ENABLED=true
74     - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
75     - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
76     - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
77     # Peer specific variabes
78     - CORE_PEER_ID=peer0.org1.example.com
79     - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
80     - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
81     - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
82     - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
83     - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
84     - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
85     - CORE_PEER_LOCALMSPID=Org1MSP
86     volumes:
87     - /var/run/:/host/var/run/
88     - ../organizations/peerOrganizations/org1.example.com/peers/peer0.org1.e
89 xample.com/msp:/etc/hyperledger/fabric/msp
90     - ../organizations/peerOrganizations/org1.example.com/peers/peer0.org1.e
91 xample.com/tls:/etc/hyperledger/fabric/tls
92     - peer0.org1.example.com:/var/hyperledger/production
93     working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
94     command: peer node start
95     ports:
96     - 7051:7051
97     networks:
98     - test
99
100     peer1.org1.example.com:
101     container_name: peer1.org1.example.com
102     image: hyperledger/fabric-peer:$IMAGE_TAG
103     environment:
104     #Generic peer variables
105     - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
106     # the following setting starts chaincode containers on the same
107     # bridge network as the peers
108     # https://docs.docker.com/compose/networking/
109     - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_test
110     - FABRIC_LOGGING_SPEC=INFO
111     #- FABRIC_LOGGING_SPEC=DEBUG
112     - CORE_PEER_TLS_ENABLED=true
113     - CORE_PEER_PROFILE_ENABLED=true
114     - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
115     - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
116     - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
117     # Peer specific variabes
118     - CORE_PEER_ID=peer1.org1.example.com

```

```

117     - CORE_PEER_ADDRESS=peer1.org1.example.com:8051
118     - CORE_PEER_LISTENADDRESS=0.0.0.0:8051
119     - CORE_PEER_CHAINCODEADDRESS=peer1.org1.example.com:8052
120     - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:8052
121     - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.org1.example.com:8051
122     - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.com:8051
123     - CORE_PEER_LOCALMSPID=Org1MSP
124     volumes:
125     - /var/run:/host/var/run/
126     - ../organizations/peerOrganizations/org1.example.com/peers/peer1.org1.e
127 xample.com/msp:/etc/hyperledger/fabric/msp
128     - ../organizations/peerOrganizations/org1.example.com/peers/peer1.org1.e
129 xample.com/tls:/etc/hyperledger/fabric/tls
130     - peer1.org1.example.com:/var/hyperledger/production
131     working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
132     command: peer node start
133     ports:
134     - 8051:8051
135     networks:
136     - test
137
138 peer0.org2.example.com:
139     container_name: peer0.org2.example.com
140     image: hyperledger/fabric-peer:$IMAGE_TAG
141     environment:
142     #Generic peer variables
143     - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
144     # the following setting starts chaincode containers on the same
145     # bridge network as the peers
146     # https://docs.docker.com/compose/networking/
147     - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_test
148     - FABRIC_LOGGING_SPEC=INFO
149     #- FABRIC_LOGGING_SPEC=DEBUG
150     - CORE_PEER_TLS_ENABLED=true
151     - CORE_PEER_PROFILE_ENABLED=true
152     - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
153     - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
154     - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
155     # Peer specific variabes
156     - CORE_PEER_ID=peer0.org2.example.com
157     - CORE_PEER_ADDRESS=peer0.org2.example.com:9051
158     - CORE_PEER_LISTENADDRESS=0.0.0.0:9051
159     - CORE_PEER_CHAINCODEADDRESS=peer0.org2.example.com:9052
160     - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:9052
161     - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org2.example.com:9051
162     - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org2.example.com:9051
163     - CORE_PEER_LOCALMSPID=Org2MSP
164     volumes:
165     - /var/run:/host/var/run/
166     - ../organizations/peerOrganizations/org2.example.com/peers/peer0.org2.e
167 xample.com/msp:/etc/hyperledger/fabric/msp
168     - ../organizations/peerOrganizations/org2.example.com/peers/peer0.org2.e
169 xample.com/tls:/etc/hyperledger/fabric/tls
170     - peer0.org2.example.com:/var/hyperledger/production
171     working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
172     command: peer node start
173     ports:
174     - 9051:9051
175     networks:
176     - test
177
178 peer1.org2.example.com:
179     container_name: peer1.org2.example.com

```



```

176 image: hyperledger/fabric-peer:$IMAGE_TAG
177 environment:
178   #Generic peer variables
179   - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
180   # the following setting starts chaincode containers on the same
181   # bridge network as the peers
182   # https://docs.docker.com/compose/networking/
183   - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_test
184   - FABRIC_LOGGING_SPEC=INFO
185   #- FABRIC_LOGGING_SPEC=DEBUG
186   - CORE_PEER_TLS_ENABLED=true
187   - CORE_PEER_PROFILE_ENABLED=true
188   - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
189   - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
190   - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
191   # Peer specific variables
192   - CORE_PEER_ID=peer1.org2.example.com
193   - CORE_PEER_ADDRESS=peer1.org2.example.com:10051
194   - CORE_PEER_LISTENADDRESS=0.0.0.0:10051
195   - CORE_PEER_CHAINCODEADDRESS=peer1.org2.example.com:10052
196   - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:10052
197   - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org2.example.com:10051
198   - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.org2.example.com:10051
199   - CORE_PEER_LOCALMSPID=Org2MSP
200 volumes:
201   - /var/run/:/host/var/run/
202   - ../organizations/peerOrganizations/org2.example.com/peers/peer1.org2.e
203 xample.com/msp:/etc/hyperledger/fabric/msp
204   - ../organizations/peerOrganizations/org2.example.com/peers/peer1.org2.e
205 xample.com/tls:/etc/hyperledger/fabric/tls
206   - peer1.org2.example.com:/var/hyperledger/production
207 working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
208 command: peer node start
209 ports:
210   - 10051:10051
211 networks:
212   - test

```

```

1  export PATH=${PWD}/../bin:$PATH
2  export FABRIC_CFG_PATH=${PWD}/config
3
4  source scriptUtils.sh
5
6  # Print the usage message
7  function printHelp() {
8      println
9      println "Usage: "
10     println " network.sh <Mode> [Flags]"
11     println " Modes:"
12     println " \033[0;32mup\033[0m – bring up fabric orderer and peer nodes. After that create and join a cha
nne;"
13     println " \033[0;32mdown\033[0m – shut down all fabric services and erase channel artifacts"
14     println " \033[0;32mdeployCC\033[0m – deploy a specified chaincode according to the input flags"
15     println
16     println " Flags: "
17     println " Used with \033[0;32mnetwork.sh up\033[0m : "
18     println " -c <channel name> – channel name to use (defaults to \"mychannel\")"
19     println " -db <dbtype> – the database backend to use: goleveldb (default) or couchdb"
20     println
21     println " Used with \033[0;32mnetwork.sh deployCC\033[0m "
22     println " -c <channel name> – deploy chaincode to channel"
23     println " -ccn <name> – the name of the chaincode to deploy"
24     println " -ccv <version> – chaincode version."
25     println " -ccp <path> – Path to the chaincode."
26     println
27     println " Used with \033[0;32mnetwork.sh testCC\033[0m "
28     println " -c <channel name> – deploy chaincode to channel"
29     println " -ccn <name> – the short name of the chaincode to deploy"
30     println
31     println " -h – print this message"
32     println
33     println " Examples:"
34     println " network.sh up -c mychannel -db couchdb"
35     println " network.sh deployCC -c mychannel -ccn contract_rawOne -ccv 1 -ccp ../chaincode/src/rawOne/
-ccl golang"
36     println " network.sh testCC -c mychannel -ccn contract_rawOne"
37     println " network.sh down"
38     println
39 }
40
41 function cleanFolder() {
42     rm -rf ./organizations
43     rm ./channel-artifacts/*
44     rm -rf ./system-genesis-block/*
45     rm log.txt
46     rm *.tar.gz
47 }
48
49 function clearContainers() {
50     CONTAINER_IDS=$(docker ps -a | awk '($2 ~ /dev-peer.*/) {print $1}')
51     if [ -z "$CONTAINER_IDS" -o "$CONTAINER_IDS" == " " ]; then
52         infoln "No containers available for deletion"
53     else
54         docker rm -f $CONTAINER_IDS
55     fi
56 }
57
58 function removeUnwantedImages() {
59     DOCKER_IMAGE_IDS=$(docker images | awk '($1 ~ /dev-peer.*/) {print $3}')
60     if [ -z "$DOCKER_IMAGE_IDS" -o "$DOCKER_IMAGE_IDS" == " " ]; then
61         infoln "No images available for deletion"

```

```

62     else
63         docker rmi -f $DOCKER_IMAGE_IDS
64     fi
65 }
66
67 # Create Organization crypto material using cryptogen or CAs
68 function createOrgs() {
69
70     if [ -d "organizations/peerOrganizations" ]; then
71         rm -Rf organizations/peerOrganizations && rm -Rf organizations/ordererOrganizations
72     fi
73
74     infoln "Generate certificates using cryptogen tool"
75
76     set -x
77     cryptogen generate --config=/config/crypto-config.yaml --output="organizations" >&log.txt
78     res=$?
79     { set +x; } 2>/dev/null
80     if [ $res -ne 0 ]; then
81         fatalln "Failed to generate certificates..."
82     fi
83 }
84
85 # Generate orderer system channel genesis block.
86 function createConsortium() {
87
88     infoln "Generating Orderer Genesis block"
89
90     set -x
91     configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/g
92     enesis.block >&log.txt
93     res=$?
94     { set +x; } 2>/dev/null
95     if [ $res -ne 0 ]; then
96         fatalln "Failed to generate orderer genesis block..."
97     fi
98 }
99
100 # Bring up the peer and orderer nodes using docker compose.
101 function networkUp() {
102     println ""
103     infoln "Bringing up the network ..."
104
105     # generate artifacts if they don't exist
106     if [ ! -d "organizations/peerOrganizations" ]; then
107         createOrgs
108         createConsortium
109     fi
110
111     COMPOSE_FILES="-f ${COMPOSE_FILE_BASE}"
112
113     if [ "${DATABASE}" == "couchdb" ]; then
114         COMPOSE_FILES="${COMPOSE_FILES} -f ${COMPOSE_FILE_COUCH}"
115     fi
116
117     IMAGE_TAG=$IMAGETAG docker-compose ${COMPOSE_FILES} up -d 2>&1
118
119     successln "Network is up and running"
120 }
121
122 ## call the script to join create the channel and join the peers of org1 and org
123 2

```

```

123 function createChannel() {
124
125     scripts/createChannel.sh $CHANNEL_NAME
126     if [ $? -ne 0 ]; then
127         fatalln "Create channel failed"
128     fi
129 }
130
131 ## Call the script to install and instantiate a chaincode on the channel
132 function deployCC() {
133
134     scripts/deployCC.sh $CHANNEL_NAME $CC_NAME $CC_SRC_LANGUAGE $CC_SRC_PATH $CC
135     _VERSION
136
137     if [ $? -ne 0 ]; then
138         fatalln "Deploying chaincode failed"
139     fi
140
141     exit 0
142 }
143
144 function networkDown() {
145     infoln "Bringing network down and cleaning the folder ..."
146     docker-compose -f $COMPOSE_FILE_BASE -f $COMPOSE_FILE_COUCH -f $COMPOSE_FILE
147     _CA down --volumes --remove-orphans
148     clearContainers
149     removeUnwantedImages
150     cleanFolder
151 }
152
153 # Default Values
154 CHANNEL_NAME="mychannel"
155 CC_NAME="contract_rawOne"
156 CC_SRC_PATH="./chaincode/src/rawOne/"
157 IMAGETAG="latest"
158
159 # use this as the default docker-compose yaml definition
160 COMPOSE_FILE_BASE=docker/docker-compose.yaml
161 # docker-compose.yaml file if you are using couchdb
162 COMPOSE_FILE_COUCH=docker/docker-compose-couch.yaml
163 # certificate authorities compose file
164 COMPOSE_FILE_CA=docker/docker-compose-ca.yaml
165 # this is the default chaincode language
166 CC_SRC_LANGUAGE="golang"
167
168 ## Parse mode
169 if [[ $# -lt 1 ]] ; then
170     printHelp
171     exit 0
172 else
173     MODE=$1
174     shift
175 fi
176
177 # parse flags
178 while [[ $# -ge 1 ]] ; do
179     key="$1"
180     case $key in
181         -h )
182             printHelp
183             exit 0
184             ;;
185         -c )
186

```

```

184         CHANNEL_NAME="$2"
185         shift
186     ;;
187     -db )
188         DATABASE="$2"
189         shift
190     ;;
191     -ccn )
192         CC_NAME="$2"
193         shift
194     ;;
195     -ccv )
196         CC_VERSION="$2"
197         shift
198     ;;
199     -ccp )
200         CC_SRC_PATH="$2"
201         shift
202     ;;
203     -ccl )
204         CC_SRC_LANGUAGE="$2"
205         shift
206     ;;
207     * )
208         errorln "Unknown flag: $key"
209         printHelp
210         exit 1
211     ;;
212 esac
213 shift
214 done
215
216 if [ "${MODE}" == "up" ]; then
217     networkUp
218     createChannel
219     elif [ "${MODE}" == "deployCC" ]; then
220         deployCC
221     elif [ "${MODE}" == "down" ]; then
222         networkDown
223     elif [ "${MODE}" == "testCC" ]; then
224         if [ $CC_NAME == "contract_rawOne" ]; then
225             scripts/testChaincode_rawOne.sh $CHANNEL_NAME $CC_NAME
226         fi
227         if [ $CC_NAME == "contract_rawTwo" ]; then
228             scripts/testChaincode_rawTwo.sh $CHANNEL_NAME $CC_NAME
229         fi
230         if [ $CC_NAME == "contract_mlModel" ]; then
231             scripts/testChaincode_mlModel.sh $CHANNEL_NAME $CC_NAME
232         fi
233         if [ $CC_NAME == "contract_mnist_mat" ]; then
234             scripts/testChaincode_mnist_mat.sh $CHANNEL_NAME $CC_NAME
235         fi
236         if [ $CC_NAME == "contract_mnist_rat" ]; then
237             scripts/testChaincode_mnist_rat.sh $CHANNEL_NAME $CC_NAME
238         fi
239     else
240         printHelp
241         exit 1
242     fi

```

```

1  #!/bin/bash
2
3  source scriptUtils.sh
4
5  CHANNEL_NAME="$1"
6  : ${CHANNEL_NAME:="mychannel"}
7
8
9  # import utils
10 . scripts/envVar.sh
11
12 if [ ! -d "channel-artifacts" ]; then
13     mkdir channel-artifacts
14 fi
15
16 createChannelTx() {
17
18     set -x
19     configtxgen -profile BasicChannel -outputCreateChannelTx ./channel-artifacts
20     /${CHANNEL_NAME}.tx -channelID $CHANNEL_NAME >&log.txt
21     res=$?
22     { set +x; } 2>/dev/null
23     if [ $res -ne 0 ]; then
24         fataln "Failed to generate channel configuration transaction..."
25     fi
26
27     successln "Success on generating channels transaction"
28 }
29
30 createAnchorPeerTx() {
31     for orgmsp in Org1MSP Org2MSP; do
32
33         set -x
34         configtxgen -profile BasicChannel -outputAnchorPeersUpdate ./channel-art
35         ifacts/${orgmsp}anchors.tx -channelID $CHANNEL_NAME -asOrg ${orgmsp} >&log.txt
36         res=$?
37         { set +x; } 2>/dev/null
38         if [ $res -ne 0 ]; then
39             fataln "Failed to generate anchor peer update transaction for ${orgmsp}..."
40         fi
41     done
42
43     successln "Success on generating anchor peer transaction"
44 }
45
46 createChannel() {
47     setGlobals 1 0
48
49     local rc=1
50     local COUNTER=1
51     local MAX_RETRY=5
52     while [ $rc -ne 0 -a $COUNTER -lt $MAX_RETRY ] ; do
53         sleep 3
54         set -x
55         peer channel create -o localhost:7050 -c $CHANNEL_NAME --ordererTLSHostn
56         ameOverride orderer.example.com -f ./channel-artifacts/${CHANNEL_NAME}.tx --outp
57         utBlock ./channel-artifacts/${CHANNEL_NAME}.block --tls --cafile $ORDERER_CA >&l
58         og.txt
59
60         res=$?
61         { set +x; } 2>/dev/null
62         let rc=$res
63         COUNTER=$((COUNTER + 1))

```

```

59     done
60
61     verifyResult $res "Channel creation failed"
62     successln "Channel '$CHANNEL_NAME' created"
63 }
64
65 joinChannel() {
66     ORG=$1
67     PEER=$2
68     setGlobals $ORG $PEER
69
70     local rc=1
71     local COUNTER=1
72     local MAX_RETRY=5
73     while [ $rc -ne 0 -a $COUNTER -lt $MAX_RETRY ] ; do
74         sleep 3
75         set -x
76         peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block >&log.txt
77         res=$?
78         { set +x; } 2>/dev/null
79         let rc=$res
80         COUNTER=$((expr $COUNTER + 1))
81     done
82
83     verifyResult $res "peer${PEER}.org${ORG} has failed to join channel '$CHANNEL_NAME' "
84     successln "peer${PEER}.org${ORG} has joined the channel '$CHANNEL_NAME'"
85 }
86
87 updateAnchorPeers() {
88     ORG=$1
89     PEER=$2
90     setGlobals $ORG $PEER
91
92     local rc=1
93     local COUNTER=1
94     local MAX_RETRY=5
95     while [ $rc -ne 0 -a $COUNTER -lt $MAX_RETRY ] ; do
96         sleep 3
97         set -x
98         peer channel update -o localhost:7050 --ordererTLSHostnameOverride order
99         er.example.com -c $CHANNEL_NAME -f ./channel-artifacts/${CORE_PEER_LOCALMSPID}an
100         chors.tx --tls --cafile $ORDERER_CA >&log.txt
101         res=$?
102         { set +x; } 2>/dev/null
103         let rc=$res
104         COUNTER=$((expr $COUNTER + 1))
105     done
106
107     verifyResult $res "Anchor peer update failed"
108     successln "Anchor peers updated for org '$CORE_PEER_LOCALMSPID' on channel '$CHANNEL_NAM
109 E'"
110 }
111
112 verifyResult() {
113     if [ $1 -ne 0 ]; then
114         fatalln "$2"
115     fi
116 }
117
118 FABRIC_CFG_PATH=${PWD}/config
119
120 ## Create channeltx

```

```
119 println ""
120 infoLn "Generating channel create transaction '${CHANNEL_NAME}.tx'"
121 createChannelTx
122
123 ## Create anchorpeertx
124 println ""
125 infoLn "Generating anchor peer update transactions"
126 createAnchorPeerTx
127
128 ## Create channel
129 println ""
130 infoLn "Creating channel ${CHANNEL_NAME}"
131 createChannel
132
133 ## Join all the peers to the channel
134 println ""
135 infoLn "Join Org1 peers to the channel..."
136 joinChannel 1 0
137 joinChannel 1 1
138 infoLn "Join Org2 peers to the channel..."
139 joinChannel 2 0
140 joinChannel 2 1
141
142 ## Set the anchor peers for each org in the channel
143 println ""
144 infoLn "Updating anchor peers for org1..."
145 updateAnchorPeers 1 0
146 infoLn "Updating anchor peers for org2..."
147 updateAnchorPeers 2 0
148
149 println ""
150 successLn "Channel successfully joined"
151
152 exit 0
```



```

1  #!/bin/bash
2
3  source scriptUtils.sh
4
5  # import utils
6  . scripts/envVar.sh
7
8  FABRIC_CFG_PATH=$PWD/config/
9
10 CHANNEL_NAME=${1:-"mychannel"}
11 CC_NAME=${2:-"rawOne"}
12 CC_SRC_LANGUAGE=${3:-"golang"}
13 CC_SRC_PATH=${4:-"/chaincode/src/rawOne/"}
14 CC_VERSION=${5:-"1.0"}
15
16 # do some language specific preparation to the chaincode before packaging
17 vendorGoDependencies() {
18     infoLn "Vendoring Go dependencies at $CC_SRC_PATH"
19     pushd $CC_SRC_PATH
20     GO11MODULE=on go mod vendor
21     popd
22     successLn "Finished vendoring Go dependencies"
23 }
24
25 packageChaincode() {
26     ORG=$1
27     PEER=$2
28     setGlobals $ORG $PEER
29
30     set -x
31     peer lifecycle chaincode package ${CC_NAME}.tar.gz --path ${CC_SRC_PATH} --l
32 ang ${CC_SRC_LANGUAGE} --label ${CC_NAME}_${CC_VERSION} >&log.txt
33     res=$?
34     { set +x; } 2>/dev/null
35     cat log.txt
36     verifyResult $res "Chaincode packaging on peer0.org${ORG} has failed"
37     successLn "Chaincode is packaged on peer0.org${ORG}"
38 }
39
40 # installChaincode PEER ORG
41 installChaincode() {
42     ORG=$1
43     PEER=$2
44     setGlobals $ORG $PEER
45
46     set -x
47     peer lifecycle chaincode install ${CC_NAME}.tar.gz >&log.txt
48     res=$?
49     { set +x; } 2>/dev/null
50
51     verifyResult $res "Chaincode installation on peer0.org${ORG} has failed"
52     successLn "Chaincode is installed on peer0.org${ORG}"
53 }
54
55 # queryInstalled PEER ORG
56 queryInstalled() {
57     ORG=$1
58     PEER=$2
59     setGlobals $ORG $PEER
60
61     set -x
62     peer lifecycle chaincode queryinstalled >&log.txt

```

```

63     res=$?
64     { set +x; } 2>/dev/null
65
66     PACKAGE_ID=$(sed -n "/${CC_NAME}_${CC_VERSION}/s/^Package ID: //; s/, Label:.*$//; p;" log.txt)
67     verifyResult $res "Query installed on peer${PEER}.org${ORG} has failed"
68     successln "Query installed successful on peer${PEER}.org${ORG} on channel"
69 }
70
71 # approveForMyOrg VERSION PEER ORG
72 approveForMyOrg() {
73     ORG=$1
74     PEER=$2
75     setGlobals $ORG $PEER
76
77     set -x
78     peer lifecycle chaincode approveformyorg -o localhost:7050 \
79     --ordererTLSHostnameOverride orderer.example.com \
80     --tls --cafile $ORDERER_CA \
81     --channelID $CHANNEL_NAME \
82     --name ${CC_NAME} \
83     --version ${CC_VERSION} \
84     --package-id ${PACKAGE_ID} \
85     --init-required \
86     --sequence ${CC_VERSION} >&log.txt
87     res=$?
88     { set +x; } 2>/dev/null
89
90     verifyResult $res "Chaincode definition approved on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME' failed"
91     successln "Chaincode definition approved on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME'"
92 }
93
94 # checkCommitReadiness VERSION PEER ORG
95 checkCommitReadiness() {
96     ORG=$1
97     PEER=$2
98     setGlobals $ORG $PEER
99
100     infoln "Checking the commit readiness of the chaincode definition on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME'..."
101
102     set -x
103     peer lifecycle chaincode checkcommitreadiness --channelID $CHANNEL_NAME \
104     --name ${CC_NAME} --version ${CC_VERSION} \
105     --sequence ${CC_VERSION} --init-required \
106     --output json >&log.txt
107     res=$?
108     { set +x; } 2>/dev/null
109
110     infoln "Checking the commit readiness of the chaincode definition successful on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME'"
111 }
112
113 commitChaincodeDefinition() {
114     parsePeerConnectionParameters $@
115     res=$?
116     verifyResult $res "Invoke transaction failed on channel '$CHANNEL_NAME' due to uneven number of peer and org parameters "
117
118     set -x
119     peer lifecycle chaincode commit -o localhost:7050 \

```

```

120 --ordererTLSHostnameOverride orderer.example.com \
121 --tls --cafile $ORDERER_CA --channelID $CHANNEL_NAME \
122 --name ${CC_NAME} \
123 $PEER_CONN_PARMS \
124 --version ${CC_VERSION} --sequence ${CC_VERSION} \
125 --init-required >&log.txt
126 res=$?
127 { set +x; } 2>/dev/null
128
129 verifyResult $res "Chaincode definition commit failed on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME' failed"
130 successln "Chaincode definition committed on channel '$CHANNEL_NAME'"
131
132 }
133
134 # queryCommitted ORG
135 queryCommitted() {
136     ORG=$1
137     PEER=0
138     setGlobals $ORG $PEER
139
140     infoln "Querying chaincode definition on peer${PEER}.org${ORG} on channel '$CHANNEL_NAME'..."
141
142     set -x
143     peer lifecycle chaincode querycommitted --channelID $CHANNEL_NAME --name ${C
144     C_NAME} >&log.txt
145     res=$?
146     { set +x; } 2>/dev/null
147 }
148
149 chaincodeInvokeInit() {
150     parsePeerConnectionParameters $@
151     res=$?
152     verifyResult $res "Invoke transaction failed on channel '$CHANNEL_NAME' due to uneven number o
153     f peer and org parameters "
154
155     # while 'peer chaincode' command can get the orderer endpoint from the
156     # peer (if join was successful), let's supply it directly as we know
157     # it using the "-o" option
158     set -x
159     fcn_call='{"function":"Init","Args":[]}'
160     infoln "invoke fcn call:${fcn_call}"
161
162     peer chaincode invoke -o localhost:7050 \
163     --ordererTLSHostnameOverride orderer.example.com \
164     --tls --cafile $ORDERER_CA -C $CHANNEL_NAME \
165     -n ${CC_NAME} $PEER_CONN_PARMS \
166     --isInit -c ${fcn_call} >&log.txt
167     res=$?
168     { set +x; } 2>/dev/null
169
170     verifyResult $res "Invoke execution on $PEERS failed "
171     successln "Invoke transaction successful on $PEERS on channel '$CHANNEL_NAME'"
172 }
173
174 ## package the chaincode
175 infoln "Packaging chaincode on Org1 Peer0"
176 packageChaincode 1 0
177
178 ## Install chaincode on peer0.org1, peer1.org1 and peer0.org2, peer1.org2
179 infoln "Installing chaincode on Org1"
180 installChaincode 1 0
181 installChaincode 1 1

```

```
180 infoln "Installing chaincode on Org2"
181 installChaincode 2 0
182 installChaincode 2 1
183
184 ## query whether the chaincode is installed
185 queryInstalled 1 0
186 queryInstalled 1 1
187 queryInstalled 2 0
188 queryInstalled 2 1
189
190 ## approve the definition for org1
191 approveForMyOrg 1 0
192
193 ## check whether the chaincode definition is ready to be committed
194 ## expect org1 to have approved and org2 not to
195 checkCommitReadiness 1 0
196 checkCommitReadiness 2 0
197
198 ## now approve also for org2
199 approveForMyOrg 2 0
200
201 ## check whether the chaincode definition is ready to be committed
202 ## expect them both to have approved
203 checkCommitReadiness 1 0
204 checkCommitReadiness 2 0
205
206 ## now that we know for sure both orgs have approved, commit the definition
207 infoln "Committing chaincode on Org1 and Org2"
208 commitChaincodeDefinition 1 2
209
210 ## query on both orgs to see that the definition committed successfully
211 queryCommitted 1
212 queryCommitted 2
213
214 # Init Chaincode
215 chaincodeInvokeInit 1 2
216
217 exit 0
```

```

1  source scriptUtils.sh
2
3  export CORE_PEER_TLS_ENABLED=true
4  export ORDERER_CA=${PWD}/organizations/ordererOrganizations/example.com/orderers
   /orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
5  export PEER0_ORG1_CA=${PWD}/organizations/peerOrganizations/org1.example.com/peers
   /peer0.org1.example.com/tls/ca.crt
6  export PEER0_ORG2_CA=${PWD}/organizations/peerOrganizations/org2.example.com/peers
   /peer0.org2.example.com/tls/ca.crt
7
8
9  # Set OrdererOrg.Admin globals
10 setOrdererGlobals() {
11     export CORE_PEER_LOCALMSPID="OrdererMSP"
12     export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/ordererOrganizations
   /example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.
   pem
13     export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/ordererOrganizations/exa
   mple.com/users/Admin@example.com/msp
14 }
15
16 # Set environment variables for the peer org
17 setGlobals() {
18     ORG=$1
19     PEER=$2
20
21     if [ $ORG -eq 1 ]; then
22
23         if [ $PEER -eq 0 ]; then
24             export CORE_PEER_LOCALMSPID="Org1MSP"
25             export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
26             export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization
   s/org1.example.com/users/Admin@org1.example.com/msp
27             export CORE_PEER_ADDRESS=localhost:7051
28         fi
29         if [ $PEER -eq 1 ]; then
30             export CORE_PEER_LOCALMSPID="Org1MSP"
31             export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
32             export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization
   s/org1.example.com/users/Admin@org1.example.com/msp
33             export CORE_PEER_ADDRESS=localhost:8051
34         fi
35     fi
36     if [ $ORG -eq 2 ]; then
37         if [ $PEER -eq 0 ]; then
38             export CORE_PEER_LOCALMSPID="Org2MSP"
39             export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG2_CA
40             export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization
   s/org2.example.com/users/Admin@org2.example.com/msp
41             export CORE_PEER_ADDRESS=localhost:9051
42         fi
43         if [ $PEER -eq 1 ]; then
44             export CORE_PEER_LOCALMSPID="Org2MSP"
45             export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG2_CA
46             export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganization
   s/org2.example.com/users/Admin@org2.example.com/msp
47             export CORE_PEER_ADDRESS=localhost:10051
48         fi
49     fi
50 }
51
52 # parsePeerConnectionParameters $@
53 # Helper function that sets the peer connection parameters for a chaincode

```

```
54 # operation
55 parsePeerConnectionParameters() {
56
57     PEER_CONN_PARMS=""
58     PEERS=""
59     while [ "$#" -gt 0 ]; do
60         setGlobals $1 0
61         PEER="peer0.org$1"
62         ## Set peer addresses
63         PEERS="$PEERS $PEER"
64         PEER_CONN_PARMS="$PEER_CONN_PARMS --peerAddresses $CORE_PEER_ADDRESS"
65         ## Set path to TLS certificate
66         TLSINFO=$(eval echo "--tlsRootCertFiles \$PEER0_ORG$1_CA")
67         PEER_CONN_PARMS="$PEER_CONN_PARMS $TLSINFO"
68         # shift by one to get to the next organization
69         shift
70     done
71     # remove leading space for output
72     PEERS="$(echo -e "$PEERS" | sed -e 's/^[:space:]*//')"
73 }
74
75 verifyResult() {
76     if [ $1 -ne 0 ]; then
77         fatalln "$2"
78     fi
79 }
80
81 parsePeerConnectionParameters 1 2
```

```
1  #!/bin/bash
2
3  source scriptUtils.sh
4
5  # import utils
6  . scripts/envVar.sh
7
8  FABRIC_CFG_PATH=$PWD/config/
9
10 CHANNEL_NAME=$1
11 CC_NAME=$2
12
13 write() {
14
15     peer chaincode invoke -o localhost:7050 \
16     --ordererTLSHostnameOverride orderer.example.com \
17     --tls --cafile $ORDERER_CA -C $CHANNEL_NAME \
18     -n ${CC_NAME} \
19     --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
20     --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
21     -c '{"function": "writeModel", "Args": ["id_1", "'"$model"'", "'"$weights"'"]}'
22
23 }
24
25
26 queryRecords(){
27
28     peer chaincode invoke -o localhost:7050 \
29     --ordererTLSHostnameOverride orderer.example.com \
30     --tls \
31     --cafile $ORDERER_CA \
32     -C $CHANNEL_NAME \
33     -n ${CC_NAME} \
34     --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
35     --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
36     -c '{"function": "queryModel", "Args": ["id_1"]}'
37 }
38
39 model=$(<../chaincode/src/mlModel/NodeJS/saved_model/model_base64)
40 weights=$(<../chaincode/src/mlModel/NodeJS/saved_model/weights_base64)
41
42 # write
43 queryRecords
```

```

1  #!/bin/bash
2
3  source scriptUtils.sh
4
5  # import utils
6  . scripts/envVar.sh
7
8  FABRIC_CFG_PATH=$PWD/config/
9
10 CHANNEL_NAME=$1
11 CC_NAME=$2
12
13 write() {
14
15     peer chaincode invoke -o localhost:7050 \
16         --ordererTLSHostnameOverride orderer.example.com \
17         --tls --cafile $ORDERER_CA -C $CHANNEL_NAME \
18         -n ${CC_NAME} \
19         --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
20         --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
21         -c '{"function": "write", "Args": ["50331648", "50331662", "8424bf520db261335d52a0b827a78538", "4000",
22         "201512200045"]}'
23 }
24
25 queryRecords(){
26
27     peer chaincode invoke -o localhost:7050 \
28         --ordererTLSHostnameOverride orderer.example.com \
29         --tls \
30         --cafile $ORDERER_CA \
31         -C $CHANNEL_NAME \
32         -n ${CC_NAME} \
33         --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
34         --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
35         -c '{"function": "queryRecords", "Args": [{"selector": {"_id": {"$eq": "50331648_50331662"}}}]}'
36 }
37
38 queryRecordsWithPagination(){
39
40     peer chaincode invoke -o localhost:7050 \
41         --ordererTLSHostnameOverride orderer.example.com \
42         --tls $CORE_PEER_TLS_ENABLED \
43         --cafile $ORDERER_CA \
44         -C $CHANNEL_NAME \
45         -n ${CC_NAME} \
46         --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
47         --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
48         -c '{"function": "queryRecordsWithPagination", "Args": [{"selector": {"timestamp": {"$gt": 0}}}, "3", "
49         "]}',
50 }
51
52 getRecordHistory(){
53     echo "history"
54     peer chaincode invoke -o localhost:7050 \
55         --ordererTLSHostnameOverride orderer.example.com \
56         --tls $CORE_PEER_TLS_ENABLED \
57         --cafile $ORDERER_CA \
58         -C $CHANNEL_NAME \
59         -n ${CC_NAME} \
60         --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \

```

```

62     --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
63     -c '{"function": "getRecordHistory", "Args": ["50331648_50331662"]}'
64 }
65
66 # write
67 # queryRecords
68 # queryRecordsWithPagination
69 # getRecordHistory

```



```
1  #!/bin/bash
2
3  C_RESET='\033[0m'
4  C_RED='\033[0;31m'
5  C_GREEN='\033[0;32m'
6  C_BLUE='\033[0;34m'
7  C_YELLOW='\033[1;33m'
8
9  # println echos string
10 function println() {
11     echo -e "$1"
12 }
13
14 # errorln echos i red color
15 function errorln() {
16     println "${C_RED}${1}${C_RESET}"
17 }
18
19 # successln echos in green color
20 function successln() {
21     println "${C_GREEN}${1}${C_RESET}"
22 }
23
24 # infoln echos in blue color
25 function infoln() {
26     println "${C_BLUE}${1}${C_RESET}"
27 }
28
29 # warnln echos in yellow color
30 function warnln() {
31     println "${C_YELLOW}${1}${C_RESET}"
32 }
33
34 # fatalln echos in red color and exits with fail status
35 function fatalln() {
36     errorln "$1"
37     exit 1
38 }
39
40 export -f errorln
41 export -f successln
42 export -f infoln
43 export -f warnln
```

```

1  import time
2  import os
3  import pandas as pd
4  import warnings
5  import tensorflow as tf
6  import numpy as np
7  # import tensorboard
8  # import datetime
9  import tensorflowjs as tfjs
10 from matplotlib import pyplot as plt
11 from sklearn.metrics import mean_squared_error
12 from lstm_predictor_keras import load_csvdata
13
14 g1 = tf.compat.v1.Graph()
15 with tf.compat.v1.Session(graph=g1) as sess:
16     warnings.filterwarnings("ignore")
17
18     # Declare the model's architecture (and other variables)
19     TIMESTEPS = 1 # with timestep 1 we can provide a (N, 1, 1) shaped matrix wi
20     th the N counters we want to predict the values for
21     RNN_LAYERS = [32, 32]
22     DENSE_LAYERS = [1]
23     SAVE_DIR = "/saved_model/model.h5"
24     INPUT_FILES_DIR = "/input/fakehdfs/nms"
25     EPOCHES = 5
26
27     # read the data
28     X, y = load_csvdata(INPUT_FILES_DIR, TIMESTEPS, seperate=False) # separate
29     train, val, test sets for X and y
30     print('-----')
31     print('train X shape', X['train'].shape)
32     print('train y shape', y['train'].shape)
33     print('val X shape', X['val'].shape)
34     print('val y shape', y['val'].shape)
35     print('test X shape', X['test'].shape)
36     print('test y shape', y['test'].shape)
37     print('-----')
38
39     # declare the model architecture
40     model = tf.keras.Sequential()
41
42     # add hidden/rnn layers
43     for cn in range(len(RNN_LAYERS)):
44         cn_name = "hidden_" + str(cn)
45         if cn == 0:
46             model.add(tf.keras.layers.LSTM(RNN_LAYERS[cn], return_sequences=True
47 , input_shape=(TIMESTEPS,
48
49         1))) # hidden layer; input_shape=(timesteps, # of features); if
50         not declared, works as well. #[batch size][timesteps][features][channels?]
51         else:
52             model.add(tf.keras.layers.LSTM(RNN_LAYERS[cn], return_sequences=True
53 , name=cn_name,
54
55             recurrent_activation="sigmoid"))
56
57     # add dense layers
58     for dn in range(len(DENSE_LAYERS)):
59         dn_name = "dense_" + str(dn)
60         model.add(tf.keras.layers.Dense(DENSE_LAYERS[dn], name=dn_name))
61
62     # compile the model
63     model.compile(optimizer='SGD', loss=tf.keras.losses.mean_squared_error)
64
65

```

```

58 # fit the model
59 model.fit(x=X['train'], y=y['train'], epochs=EPOCHES, verbose=0)
60
61 # predict using the model
62 start = time.time()
63 for i in range(8000):
64     predicted = np.array(model.predict(X['test'], verbose=0))
65 end = time.time()
66 print(end - start)
67
68 # reshape to get the same shape as y['test'] for correct calculations
69 # predicted = predicted.reshape((predicted.shape[0], predicted.shape[2]))
70 #
71 # print('predicted shape', predicted.shape)
72 # print('-----')
73
74 # # calculate evaluation metrics
75 # difference = np.subtract(predicted, y['test'])
76 # rmse = np.sqrt((np.asarray(difference) ** 2).mean())
77 # score = mean_squared_error(predicted, y['test'])
78 # nmse = score / np.var(y['test'])
79 #
80 # # print evaluation metrics
81 # print("RSME: %f" % rmse) # Root Mean Square Error
82 # print("NSME: %f" % nmse) # Normalized Mean Square Error
83 # print("MSE: %f" % score) # Mean Square Error
84 #
85 # # Visualize
86 # plot_test, = plt.plot(y['test'], label='test')
87 # plot_predicted, = plt.plot(predicted, label='predicted')
88 # plt.legend(handles=[plot_predicted, plot_test])
89 # plt.show()
90
91 # Export into TF.js Layers format
92 # tfjs.converters.save_keras_model(model, SAVE_DIR)
93
94 # Export- Keras model Approach
95
96 # Keras exporting method
97 # export the full model to file system
98 # model.save(SAVE_DIR)
99
100 # tfjs.converters.save_keras_model(model, SAVE_DIR)
101
102 #import the model
103 # model = tf.keras.models.load_model(SAVE_DIR)
104
105 #*****
106 # predict using the model
107 # predicted = np.array(model.predict(X['test'], verbose=0))
108 #
109 # # reshape to get the same shape as y['test'] for correct calculations
110 # predicted = predicted.reshape((predicted.shape[0], predicted.shape[1]))
111 #
112 # print('predicted shape', predicted.shape)
113 # print('-----')
114 #
115 # # calculate evaluation metrics
116 # difference = np.subtract(predicted, y['test'])
117 # rmse = np.sqrt((np.asarray(difference) ** 2).mean())
118 # score = mean_squared_error(predicted, y['test'])
119 # nmse = score / np.var(y['test'])
120 #

```

```

121 # # print evaluation metrics
122 # print("RSME: %f" % rmse) # Root Mean Square Error
123 # print("NSME: %f" % nmse) # Normalized Mean Square Error
124 # print("MSE: %f" % score) # Mean Square Error
125 #
126 # # Visualize
127 # plot_test, = plt.plot(y['test'], label='test')
128 # plot_predicted, = plt.plot(predicted, label='predicted')
129 # plt.legend(handles=[plot_predicted, plot_test])
130 # plt.show()
131
132 # *****

```

```

1  import os
2  import numpy as np
3  import pandas as pd
4
5  import warnings
6
7  warnings.filterwarnings("ignore")
8
9
10 def rnn_data(data, time_steps, labels=False):
11     """
12     """
13     Creates all the possible sub-arrays (consisting of consequent array elements) of size "time_steps".
14     This means that the model will 'learn' this sequences, and we can hence use the first element of the
15     time_steps-long sequence to predict/remember the last one.
16     """
17     creates new data frame based on previous observation
18     * example:
19     l = [1,2,3,4,5]
20     time_steps = 2
21     -> labels == False [[1, 2], [2, 3], [3, 4]] #Data frame for input with 2 timesteps
22     -> labels == True [3, 4, 5] # labels for predicting the next timestep
23     """
24     rnn_df = []
25     for i in range(len(data) - time_steps):
26         if labels:
27             try:
28                 rnn_df.append(data.iloc[i + time_steps].to_numpy())
29             except AttributeError:
30                 rnn_df.append(data.iloc[i + time_steps])
31         else:
32             data_ = data.iloc[i: i + time_steps].to_numpy()
33             rnn_df.append(data_ if len(data_.shape) > 1 else [[i] for i in data_
34 ])
35
36     return np.array(rnn_df, dtype=np.float32)
37
38 def split_data(data, val_size=0.1, test_size=0.1):
39     """
40     splits data to training, validation and testing parts; defaulting in 80-10-10
41     """
42     ntest = int(round(len(data) * (1 - test_size)))
43     nval = int(round(len(data.iloc[:ntest]) * (1 - val_size)))
44     df_train, df_val, df_test = data.iloc[:nval], data.iloc[nval:ntest], data.iloc[:ntest]
45
46     return df_train, df_val, df_test
47
48 def prepare_data(data, time_steps, labels=False, val_size=0.1, test_size=0.5):
49     """
50     Given the number of 'time_steps' and some data,
51     prepares training, validation and test data for an lstm cell. (lstm cells use ordinary rnn data)
52     """
53     df_train, df_val, df_test = split_data(data, val_size, test_size)
54     return (rnn_data(df_train, time_steps, labels=labels),
55             rnn_data(df_val, time_steps, labels=labels),
56             rnn_data(df_test, time_steps, labels=labels))
57
58 def read_input_files(working_dir):
59     """
60     #Read the data from the input files (column indexing starts from zero)
61     #time -> This is a timestamp of the record.

```

```

62     #meas_info -> This is the site of the cell tower (antenna).
63     #     You can think of this as a part of the telco's antenna with a specific range.
64     #counter -> This is a number that represents a predefined KPI (Key Performance Indicator).
65     #value -> Is the value of the KPI.
66     :param working_dir: directory in which the input files are in
67     :return: rawdata
68     '''
69     rawdata = None
70     df_list = []
71     for root, dirs, files in os.walk(working_dir):
72         file_list = []
73
74         for filename in files:
75             if filename.endswith('.csv'):
76                 file_list.append(os.path.join(root, filename))
77         for file in file_list:
78             df = pd.read_csv(file, delimiter="|", usecols=[1, 2, 6, 7], header=None, na_values=["NIL"],
79                             na_filter=True, names=["meas_info", "counter", "value", "time"], index_col='time')
80             df = df[df["counter"] == 67194794] # extract data for this counter only
81             df_list.append(df[["value"]])
82
83     if df_list:
84         rawdata = pd.concat(df_list)
85
86     return rawdata
87
88
89 def load_csvdata(working_dir, time_steps, separate=False):
90     '''Separates the data into train, validation and test sets, and returns them'''
91
92     data = read_input_files(working_dir)
93     if not isinstance(data, pd.DataFrame):
94         data = pd.DataFrame(data)
95     train_x, val_x, test_x = prepare_data(data['a'] if separate else data, time_steps)
96     train_y, val_y, test_y = prepare_data(data['b'] if separate else data, time_steps, labels=True)
97     return dict(train=train_x, val=val_x, test=test_x), dict(train=train_y, val=val_y, test=test_y)
98
99
100 def read_input_files_cell(working_dir, cell_name):
101     '''
102     #Read the data from the input files (column indexing starts from zero)
103     #meas_info -> This is the site of the cell tower (antenna).
104     #     You can think of this as a part of the telco's antenna with a specific range.
105     #cell_name -> The name of the cell tower.
106     #counter -> This is a number that represents a predefined KPI (Key Performance Indicator).
107     #value -> Is the value of the KPI.
108     #time -> This is a timestamp of the record.
109     :param working_dir: the directory of the input files
110     :param cell_name: the name of the cell for whose data will be retrieved
111     :return: rawdata
112     '''
113     rawdata = None
114     df_list = []
115     for root, dirs, files in os.walk(working_dir):
116         file_list = []
117
118         for filename in files:
119             if filename.endswith('.csv'):
120                 file_list.append(os.path.join(root, filename))

```

```

121     for file in file_list:
122         df = pd.read_csv(file, delimiter="|", usecols=[1, 2, 5, 6, 7], header=None, na_values=["NIL"],
123             na_filter=True, names=["meas_info", "counter", "cellname", "value", "time"],
124             index_col='time')
125         df = df[df["cellname"].str.contains(cell_name)]
126         if not df.empty:
127             df_list.append(df[["value"]])
128
129     if df_list:
130         rawdata = pd.concat(df_list)
131
132
133     return rawdata
134
135
136
137
138 def load_csvdata_cell(working_dir, cell_name, time_steps, seporate=False):
139     '''Separates the data into train, validation and test sets, and returns them'''
140
141     data = read_input_files_cell(working_dir, cell_name)
142     if not isinstance(data, pd.DataFrame):
143         data = pd.DataFrame(data)
144     train_x, val_x, test_x = prepare_data(data['a'] if seporate else data, time_steps)
145     train_y, val_y, test_y = prepare_data(data['b'] if seporate else data, time_steps, labels=True)
146     return dict(train=train_x, val=val_x, test=test_x), dict(train=train_y, val=val_y, test=test_y)
147
148
149
150 def find_all_cell_names(working_dir):
151     for root, dirs, files in os.walk(working_dir):
152         file_list = []
153
154         for filename in files:
155             if filename.endswith('.csv'):
156                 file_list.append(os.path.join(root, filename))
157         df_cells_list = []
158         for file in file_list:
159             df_cells = pd.read_csv(file, delimiter="|", usecols=[5, 7], header=None, na_values=["NIL"],
160                 na_filter=True, names=["cellname", "time"], index_col='time')
161             df_cells_list.append(df_cells["cellname"].tolist())
162
163         if df_cells_list:
164             cells = np.unique(df_cells_list)
165
166     return cells

```