ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο υπολογιστικών συστημάτων

Επιβεβαιώσιμη Κατανεμημένη Επεξεργασία στο Apache Spark με την βοήθεια Smart Contract

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Κωνσταντίνου Σ.
Μαυρογεώργη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο υπολογιστικών συστημάτων

Επιβεβαιώσιμη Κατανεμημένη Επεξεργασία στο Apache Spark με την βοήθεια Smart Contract

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Κωνσταντίνου Σ.
Μαυρογεώργη

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Πέμπτη 2/3/2023.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Δημήτριος Τσουμάκος
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023.

.....
(Κωνσταντίνος Σ. Μαυρογεώργης)
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η συνεχής αύξηση τους όγκου δεδομένων προς επεξεργασία έχει οδηγήσει στην δημιουργία ανοιχτών clusters, στους οποίους μπορεί να συμμετέχει όποιος το επιθυμεί είτε εθελοντικά είτε για κάποια αμοιβή, με σκοπό την αύξηση της κλιμακωσιμότητας και της υπολογιστικής ισχύος. Σε αυτούς τους clusters, οι κόμβοι που συμμετέχουν δεν είναι έμπιστοι και δεν υπάρχει εγγύηση ότι θα εκτελέσουν σωστά τις εργασίες που τους ανατίθενται. Επομένως, σκοπός της παρούσας εργασίας είναι η μελέτη και η υλοποίηση ενός συστήματος επιβεβαιώσιμης κατανεμημένης επεξεργασίας δεδομένων το οποίο με την βοήθεια ενός έξυπνου συμβολαίου παρέχει πιστοποίηση τόσο στον χρήστη όσο και στην συστάδα των υπολογιστών-κόμβων ότι τα αποτελέσματα των εργασιών είναι σωστά και κανένα εμπλεκόμενο μέλος δεν προσπαθεί να εκμεταλλευτεί το σύστημα προς όφελός του. Σε αυτό το σύστημα, το έξυπνο συμβόλαιο λειτουργεί ως έμπιστη αρχή που πραγματοποιεί τους απαραίτητους ελέγχους για να επιβεβαιώσει την επεξεργασία. Για την συγκεκριμένη πτυχιακή εργασία θα επεκταθεί το framework Apache Spark, το οποίο είναι ένα από τα γνωστότερα εργαλεία κατανεμημένης εργασίας ανοιχτού κώδικα.

Λέξεις Κλειδιά

Apache Spark, Κατανεμημένα Συστήματα, Επιβεβαιώσιμη Επεξεργασία, Smart Contract, Αντιγραφή Tasks

Abstract

The continuous increase in the volume of data that requires processing has led to the creation of open clusters, in which anyone can participate either voluntarily or for a reward, in order to increase scalability and computing power. In these clusters, participating nodes are not trusted and there is no guarantee that they will perform the tasks assigned to them correctly. Therefore, the purpose of this thesis is to design and implement a verifiable distributed data processing system which, with the help of a smart contract, can assure both the user and the cluster that the results of the tasks are correct and no member involved tried to exploit the system to his advantage. In this system, the smart contract acts as a trusted authority that performs the necessary checks to verify the results. For this thesis, the Apache Spark framework, which is one of the most well-known open source distributed work tools, will be extended.

Keywords

Apache Spark, Distributed Systems, Verifiable computing, Smart Contract, Task Replication

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Νεκτάριο Κοζύρη που μου ανέθεσε ένα τόσο ενδιαφέρον θέμα και την κα. Κατερίνα Δόκα για την πολύτιμη συνεργασία, την διαρκή καθοδήγηση και επίβλεψη. Θα ήθελα επίσης να ευχαριστήσω τους γονείς μου, Όλγα και Στέλιο, την γιαγιά μου Δέσποινα και την υπόλοιπη οικογένεια μου για την διαρκή υποστήριξη όλα αυτά τα χρόνια. Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου, οι οποίοι με την παρέα τους αποτέλεσαν πολύ σημαντικό στηρίγματα στην καθημερινότητα μου.

Περιεχόμενα

Λίστα Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός

Τα τελευταία χρόνια, ο ρυθμός παραγωγής νέων δεδομένων που χρειάζονται επεξεργασία αυξάνεται ολοένα και γρηγορότερα. Επομένως, η κατανεμημένη επεξεργασία είναι πιο επίκαιρη από ποτέ. Κατανεμημένη επεξεργασία είναι μια ευρέως διαδεδομένη μεθοδολογία που επιτρέπει την διαίρεση μιας χρονοβόρας εργασίας σε μικρότερα tasks προκειμένου να εκτελεστούν παράλληλα σε κάποια συστάδα υπολογιστών(cluster) μειώνοντας έτσι τον χρόνο εκτέλεσης. Σε περίπτωση ανάγκης περισσότερων πόρων, συστήματα cluster καθιστούν εύκολη την κλιμάκωση με την εισαγωγή νέων μηχανημάτων στην συστάδα τους. Υπάρχουν αρκετά frameworks που χρησιμοποιούνται για κατανεμημένη επεξεργασία. Τα δύο γνωστότερα είναι το Hadoop MapReduce και το Apache Spark.

Οι περισσότεροι clusters που τρέχουν κάποιο framework κατανεμημένης επεξεργασία διαχειρίζονται από κάποιον οργανισμό ή επιχείρηση που λειτουργούν σαν έμπιστες αρχές και μπορούν να εγγυηθούν ότι όλα τα μηχανήματα της συστάδας δρουν τίμια με σκοπό την σωστή εκτέλεση των εργασιών. Σε αυτήν την περίπτωση η κλιμακωσιμότητα των συστημάτων είναι περιορισμένη από τους πόρους του οργανισμού που διαχειρίζεται τον cluster. Ωστόσο, λόγω της συνεχής αύξησης του όγκου δεδομένων προς επεξεργασία, υπάρχει διαρκής ανάγκη για αύξηση των πόρων του cluster και κατά συνέπεια της κλιμακωσιμότητας. Για τον παραπάνω λόγο, έχουν εμφανιστεί ανοιχτά clusters, στα οποία συνεισφέρει όποιος το επιθυμεί είτε εθελοντικά, είτε για κάποιο αντίτιμο, είτε για το δικαίωμα χρήσης τους cluster. Τέτοιοι selectlanguageenglishclusters έχουν αυξημένη ικανότητα κλιμακωσιμότητας και μπορούν να λύσουν τα προβλήματα των κλειστών cluster. Παρόλα αυτά, στους ανοιχτούς clusters, επειδή συμμετέχει όποιος το επιθυμεί, δεν υπάρχει κάποια αρχή να εγγυηθεί ότι

τα μηχανήματα της συστάδας είναι έμπιστα.

Επομένως, σκοπός της παρούσας διπλωματικής είναι η υλοποίηση ενός συστήματος κατανεμημένης επεξεργασίας που επεκτείνει το Apache Spark ώστε να μπορεί να τρέξει με αξιοπιστία σε ανοιχτούς clusters, των οποίων οι κόμβοι δεν είναι έμπιστοι.

1.2 Επιλογή μεθοδολογίας

Υπάρχουν πολλές μεθοδολογίες που μπορούν να συνεισφέρουν στην υλοποίηση ενός αξιόπιστους spark. Κάποιες από αυτές είναι το Truebit και το zk-Snarks. Ωστόσο, πολλές από αυτές της μεθοδολογίες έχουν αρκετούς περιορισμούς που καθιστούν δύσκολη την εφαρμογή τους πάνω σε ένα ήδη υλοποιημένο framework όπως το Apache Spark. Για αυτόν τον λόγο, στα πλαίσια αυτής την διπλωματικής εργασία θα χρησιμοποιηθεί μια πιο απλή μέθοδος.Συγκεκριμένα, θα χρησιμοποιηθεί replication των εργασιών που τρέχει το spark ώστε να εκτελούνται πάνω από μία φορά στον cluster. Με αυτόν τον τρόπο θα μπορεί να συγκριθούν τα αποτελέσματα των αντιγραμένων εργασιών ώστε να βρεθεί αν έχει συμβεί κάποιο λάθος. Πρέπει να σημειωθεί όμως ότι ο έλεγχος των αποτελεσμάτων πρέπει να γίνεται από κάποιο έμπιστο μέλος για να μπορεί να εγγυηθεί την ορθή εκτέλεση της εργασίας. Ωστόσο, σε ένα ανοιχτό cluster δεν υπάρχει έμπιστο μέλος. Για αυτό το λόγο, θα χρησιμοποιηθεί η τεχνολογία των smart contract προκειμένου ο έλεγχος των αποτελεσμάτων να γίνεται αποκεντρωμένα χωρίς την ανάγκη έμπιστης αρχής. Επιπλέον, θα μελετηθούν οι συνέπειες που προκαλεί η αλλαγή του αριθμού replication των tasks τόσο στο χρόνο εκτέλεσης όσο και στην ανοχή σε σφάλματα.

1.3 Διαχωρισμός της Διπλωματικής

Η παρούσα πτυχιακή εργασία χωρίζεται σε 7 κεφάλαια. Το κεφάλαιο 1 αποτελεί εισαγωγή στο θέμα. Στα κεφάλαια 2 και 3 παρουσιάζονται οι τεχνολογίες του Apache spark και των Smart Contract αντίστοιχα. Στο κεφάλαιο 4, παρουσιάζονται παράγοντες που μπορούν να αλλοιώσουν τα αποτελέσματα του Apache Spark. Έπειτα, στο κεφάλαιο 5 αναλύονται η αρχιτεκτονική της επέκτασης του Apache Spark. Στη συνέχεια, ακολουθεί πειραματική αξιολόγηση στο κεφάλαιο 6. Τέλος, στο κεφάλαιο 7 ακολουθούν ακολουθεί το συμπέρασμα και οι μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Apache Spark

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί ανάλυση της αρχιτεκτονικής και του τρόπου λειτουργίας του Apache Spark[1][2], το οποίο είναι και το κεντρικό αντικείμενο μελέτης της συγκεκριμένης Διπλωματικής Εργασίας.

2.1 Εισαγωγή στο Spark

Το Spark αποτελεί ένα εργαλείο ανοιχτού κώδικα που χρησιμοποιεί κατανεμημένη επεξεργασία για την εκτέλεση εφαρμογών μεγάλου όγκου δεδομένων και μπορεί να τρέξει είτε σε ένα μηχάνημα είτε σε κάποιο cluster. Μπορεί να χρησιμοποιηθεί για την επίλυση προβλημάτων όπως ETL, SQL analytics, Data science at scale και Machine Learning. Σε αντίθεση με άλλα frameworks, όπως το hadoop που χρησιμοποιεί επεξεργασία δύο σταδίων κατευθείαν στον δίσκο με MapReduce, το spark επιτρέπει σε μια εργασία να έχει περισσότερα στάδια, με την επεξεργασία να γίνεται στην μνήμη. Με αυτόν τον τρόπο καταφέρνει να αυξήσει τον αριθμό των σταδίων που τρέχουν παράλληλα. Το Spark είναι υλοποιημένο σε Scala, η οποία αποτελεί μια συναρτησιακή και αντικειμενοστραφής γλώσσα υψηλού επιπέδου που έχει όμοιο data typing με την Java και τρέχει πάνω σε JVM. Τέλος, αξίζει να σημειωθεί ότι το spark αναγνωρίζει πολλές γλώσσες προγραμματισμού όπως Scala, Python, Java, R και SQL.

2.2 Data στο Spark

RDD - Description of Distributed Computation

Το RDD είναι η βασικότερη δομή που χρησιμοποιεί το spark για την αποθήκευση, την μεταφορά και την επεξεργασία δεδομένων. Ουσιαστικά πρόκειται για μια αφαιρετική δομή από ανθεκτικά σε σφάλματα κατανεμημένα σύνολα δεδομένων που αποτελούν απλές περιγραφές υπολογισμών σε μια κατανεμημένη συλλογή εγγράφων. Αυτά τα κατανεμημένα σύνολα δεδομένων αποθηκεύονται σε διάφορους κόμβους του cluster και επιτρέπουν την παράλληλη επεξεργασία. Επιπλέον, τα δεδομένα μέσα στο RDD δεν έχουν κάποια συγκεκριμένη δομή.

Dataframe - Dataset

Τα dataframe και τα dataset αποτελούν διαφορετικές δομές απο τα RDD με κυριότερη διαφορά την δόμηση των δεδομένων σε στήλες. Από το spark 2.0 και μετά, για την μείωση της πολυπλοκότητας πραγματοποιήθηκε merging μεταξύ των dataframes και των datasets σε Strongly-Typed API και Untyped API Datasets. Αν και καθιστούν τον προγραμματισμό δυσκολότερο και πιο αργό σε ορισμένες περιπτώσεις, αξίζει να χρησιμοποιηθούν όταν εκτελείται κώδικας SQL ή όταν η δομή των δεδομένων χρειάζεται προστασία.

2.3 Τύποι πράξεων στο Spark

Το spark διαχωρίζει τις πράξεις που πραγματοποιεί στα δεδομένα του σε δύο κατηγορίες:

- Transformation(μετασχηματισμός): Ένας μετασχηματισμός είναι μια πράξη οκνηρής αποτίμησης που δέχεται ένα RDD σαν input και παράγει σαν output ένα ή περισσότερα RDDs πραγματοποιώντας τον υπολογισμό που αντιπροσωπεύει. Κάποια παραδείγματα τέτοιων πράξεων είναι τα map(), reduceByKey() και join().

- Action(δράση): Μια δράση είναι μια πράξη πάνω σε ένα rdd που παράγει συγκεκριμένες τιμές που δεν είναι σε μορφή rdd. Με άλλα λόγια, για να "σπάσει" το laziness του spark και να πραγματοποιηθεί η αποτίμηση των εκφράσεων πρέπει να χρησιμοποιηθεί ένα action. Κάποια παραδείγματα τέτοιων πράξεων είναι τα collect(), take().

2.4 Διαίρεση της Εργασίας

Για την εκτέλεση μιας εργασίας το spark διαιρεί την εργασία σε μικρότερα μέρη πριν πραγματοποιήσει την ανάθεση στους κόμβους του cluster. Αυτή η διαίρεση

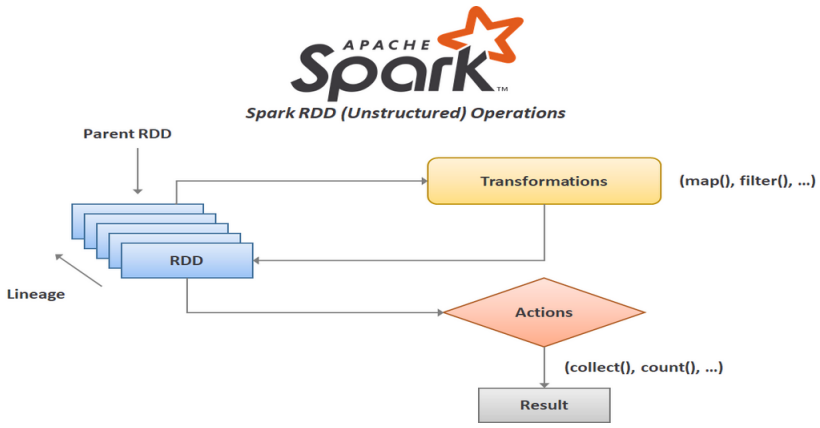
αποτελείται από τέσσερα στάδια που ξεκινάνε από το πιο γενικό και καταλήγουν στο στάδιο που θα εκτελεστεί στους κόμβους του cluster. Τα στάδια αυτά είναι:

- **App** (εφαρμογή): Η εφαρμογή αποτελεί την πιο γενική και ευρεία έννοια. Πρόκειται ουσιαστικά για την εργασία που καταθέτει κάποιος στον spark cluster μέσω spark submit. Μια εφαρμογή χωρίζεται σε jobs.
- **Job** (δουλειά): Μια δουλειά προκύπτει από το κάλεσμα ενός action μέσα σε ένα app. Επομένως, ένα app μπορεί να έχει πολλά jobs. Κάθε job στέλνεται σε μια διεργασία που ονομάζεται DAGScheduler η οποία την μετατρέπει σε έναν ακυκλικό γράφο και την χωρίζει σε stages.
- **Stage** (Στάδιο): Κάθε job χωρίζεται σε εξαρτώμενα μεταξύ τους stages. Αυτές οι εξαρτήσεις προκύπτουν από τον ακυκλικό γράφο του job και εκφράζουν ποια stages πρέπει να χρησιμοποιήσουν αποτελέσματα άλλων stages και κατά συνέπεια πρέπει να τρέξουν αργότερα χρονικά. Επομένως, μέσω των εξαρτήσεων γίνεται σαφές πια stages πρέπει να περιμένουν άλλα stages και ποια μπορούν να τρέξουν παράλληλα. Επιπλέον, αξίζει να σημειωθεί ότι η διαδικασία διαχωρισμού σε στάδια λαμβάνει υπόψη και το shuffling. Shuffling είναι η διαδικασία μεταφοράς δεδομένων στους κατάλληλους κόμβους όταν γίνεται κλήση reduceByKey(). Ένα stage συνδέεται με ένα taskset το οποίο είναι ένα σύνολο από tasks.Τα stages χωρίζονται στις εξής υποκατηγορίες:
  - ShuffleMapStage: Αυτά τα stages αποτελούν την ενδιάμεση επεξεργασία και δεν παράγουν το τελικό αποτέλεσμα
  - ResultStage: Αυτά τα stages παράγουν το τελικό αποτέλεσμα.
- **Task** (Εργασία): Ένα task αποτελεί την μικρότερη μονάδα εκτέλεσης της κάθε εφαρμογής. Για το ίδιο stage, το κάθε task είναι η ίδια εργασία σε διαφορετικό partition του rdd. Τα tasks στέλνονται στους worker κόμβους του cluster προκειμένου να τρέξουν.

Κάθε App, Job, Stage και Task έχει ένα μοναδικό αναγνωριστικό AppId, JobId, StageId και TaskId. Ωστόσο, το κάθε task έχει και ένα δεύτερο αναγνωριστικό που ονομάζεται index και δείχνει την θέση του στο taskset που ανήκει.

2.5 Αρχιτεκτονική Apache Spark

Το Spark χρησιμοποιεί αρχιτεκτονική master-slave. Αρχικά, δημιουργείται μια διεργασία που ονομάζεται spark-driver και επικοινωνεί με την διεργασία spark-master, η οποία διαχειρίζεται ένα σύνολο διεργασιών που ονομάζονται spark-workers. Οι



Σχήμα 2.1: Τύποι πράξεων στο Spark

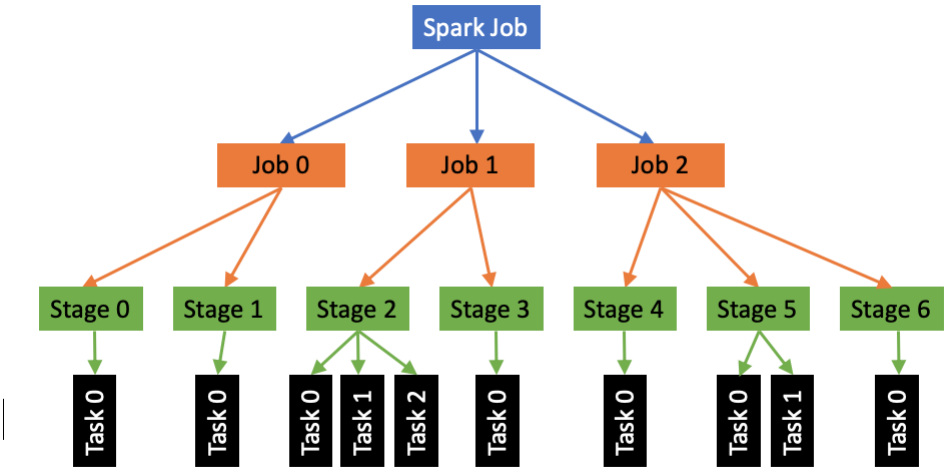
παραπάνω διεργασίες τρέχουν παράλληλα με άλλες που θα παρουσιαστούν στην συνέχεια. Για την διευκόλυνση της ανάλυσης θα θεωρηθεί ότι υπάρχει ένας χρήστη που υποβάλει εργασίες στο spark, ο οποίος θα ονομάζεται client, και ένας cluster από κόμβους που είναι υπεύθυνοι για εκτέλεση της επεξεργασίας. Στην συνέχεια, θα αναλυθούν οι σημαντικότερες διεργασίες, ο ρόλος τους, αν τρέχουν στο cluster ή στον client, καθώς και η επικοινωνία μεταξύ τους.

2.5.1 Spark Master

Ο spark master είναι μια διεργασία που τρέχει σε κάποιον κόμβο του cluster και είναι υπεύθυνη να γνωρίζει την κατάσταση των κόμβων του cluster. Επιπλέον, μέσω του master γίνεται η εισαγωγή ή η αποχώρηση κόμβων από τον cluster. Κατά την διάρκεια εκτέλεσης μια εφαρμογής, ο master, γνωρίζοντας την κατάσταση των κόμβων, αναλαμβάνει την διάθεση πόρων στον spark driver προκειμένου να ολοκληρωθεί η εκάστοτε εργασία.

2.5.2 Spark Worker

Ο spark worker είναι η διεργασία που πρέπει να τρέχει κάθε μηχάνημα που επιθυμεί να συνδεθεί στον cluster. Συγκεκριμένα, κάθε φορά που ένα μηχάνημα-κόμβος προσπαθεί να συνδεθεί στον cluster, επικοινωνεί με το spark master και ξεκινάει έναν μοναδικό spark worker. Κατά την εκτέλεση ενός app ο κάθε spark worker δημιουργεί μερικά threads, τους executors. Ο executor είναι τελικά ο πόρος που δεσμεύεται και εκτελεί τα tasks, για να το καταφέρει αυτό διαθέτει cores και μνήμη του μηχανήματος του worker. Οι executors έχουν χρόνο ζωής όσο το app. Με άλλα λόγια, ο spark



Σχήμα 2.2: Διαίρεση της Εργασίας

master κατά την διάρκεια εκτέλεσης μιας εφαρμογής, διαθέτει μη δεσμευμένους executors στον spark driver προκειμένου να τρέξουν tasks. Κάθε executor μπορεί να τρέξει ένα task την φορά και μόλις ολοκληρώσει την επεξεργασία του αποδεσμεύεται. Κάθε spark worker μπορεί να έχει διαφορετικό αριθμό από executors ανάλογα με την υπολογιστική ισχύ που διαθέτει και την μνήμη του.

2.5.3 Spark Driver

Ο Spark-driver αποτελεί την πρώτη διεργασία που δημιουργείται μόλις ο client κάνει submit κάποια δουλειά στο spark. Έχει σημασία να σημειωθεί ότι ο spark-driver αποτελεί ουσιαστικά τον master node της αρχιτεκτονικής master-slave, και δεν πρέπει να συγχέεται με τον spark master, ο οποίος έχει διαφορετικό ρόλο. Επιπλέον, ο spark-driver μπορεί να τρέξει είτε στον client είτε σε κάποιον τυχαίο worker του cluster, η επιλογή γίνεται όταν πραγματοποιείται το submit με την επιλογή deploy-mode. Επίσης, είναι υπεύθυνος για την αρχικοποίηση της εργασίας, την μετατροπή του κώδικα σε ένα πλάνο εκτέλεσης και τον διαμοιρασμό των tasks στους executors του cluster. Για να επιτευχθούν τα παραπάνω, στον spark-driver αρχικοποιούνται οι διεργασίες SparkContext, DAGScheduler και TaskScheduler, οι οποίες τρέχουν όπου τρέχει και ο driver.

SparkContext

Κάθε φορά που ο χρήστης υποβάλει μια εργασία στο spark, αυτό δημιουργεί ένα app. Κάθε τέτοιο app συνδέεται με ένα sparkContext το οποίο είναι η σημαντικότερη

διεργασία του συγκεκριμένου app, καθώς κρατάει εσωτερικές μεταβλητές για την κατάσταση του app και αποτελεί σημείο αναφοράς για τις υπόλοιπες διεργασίες. Κατά την δημιουργία του, το sparkContext, δημιουργεί σύνδεση με τον spark master. Στην συνέχεια, δημιουργεί τις διεργασίες DAGScheduler και TaskScheduler

DAGScheduler

Ο DAGScheduler είναι η διεργασία που δημιουργεί ακυκλικούς γράφους από jobs. Συγκεκριμένα, διαθέτει την μέθοδο submitJob(), με την οποία ο sparkContext στέλνει νέα jobs στον DAGScheduler. Στην συνέχεια, ο scheduler υπολογίζει τον ακυκλικό γράφο που αντιστοιχεί στο λογικό πλάνο εκτέλεσης της εφαρμογής. Ο υπολογισμός τους ακυκλικού γράφου ξεκινάει από το τελευταίο stage και υπολογίζεται αναδρομικά πηγαίνοντας στα προηγούμενα stages. Ταυτόχρονα υπολογίζει τις εξαρτήσεις μεταξύ των stages και μόλις εντοπίσει ένα stage χωρίς εξαρτήσεις σταματάει την αναδρομή και ολοκληρώνει την διαδικασία. Με αυτόν τον τρόπο, ο DAGScheduler υπολογίζει όλα τα stages που πρέπει να εκτελεστούν. Τελικά, για κάθε stage δημιουργεί ένα taskset το οποίο περιέχει τα tasks που πρέπει να τρέξουν φυσικά σε κάποιον executor του cluster. Κατά την δημιουργία του κάθε taskset δημιουργείτε και έναν taskset manager που στην συνέχεια θα βοηθήσει τον taskScheduler στην δρομολόγηση των tasks στους executors.

TaskScheduler

Ο taskScheduler είναι η διεργασία υπεύθυνη για την ανάθεση των task ενός taskset στους executors των workers προκειμένου να εκτελεστούν. Για να πραγματοποιηθεί αυτό ο taskScheduler επικοινωνεί με τον master και ζητάει διαθέσιμους executors. Έπειτα, ξεκινάει η διαδικασία αποστολής των tasks στους διαθέσιμους executors. Σε περίπτωση που δεν υπάρχουν διαθέσιμοι executors ο taskScheduler αναμένει και ο master τον ενημερώνει όταν απελευθερωθούν νέοι πόροι. Αξίζει να σημειωθεί ότι το κάθε task στέλνεται στον executor από την μέθοδο resourceOffer() του taskset manager. Η resourceOffer() καλείται από την μέθοδο του taskScheduler resourceOffers() η οποία καλείται από τον driver σαν απάντηση στους πόρους που προσφέρει ο master.

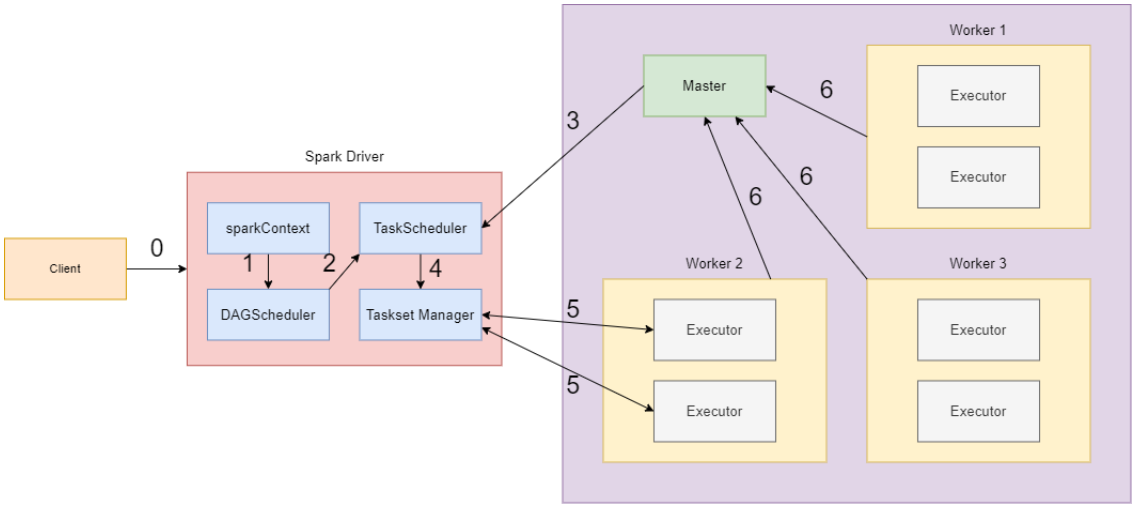
2.6 Ροή εκτέλεσης

Συνοψίζοντας, μια εφαρμογή ξεκινάει όταν ο χρήστης κάνει spark-submit την εργασία που θέλει στο spark. Στην συνέχεια, ο spark driver δημιουργεί το sparkContext που με την σειρά του δημιουργεί τον DAGScheduler και taskScheduler. Έπειτα, ο DAGScheduler δημιουργεί του ακυκλικούς γράφους και τα stages της εφαρμογής και αναθέτει στον taskScheduler να δρομολογήσει τα taskset των stages. Ο taskScheduler επικοινωνεί με τον spark master για να βρει τους διαθέσιμους executors και στην

συνέχεια με την βοήθεια του taskset manager στέλνει τα tasks στους executors των workers του cluster. Τελικά, οι executors επιστρέφουν τις απαντήσεις στον driver και η διαδικασία επαναλαμβάνεται μέχρι να ολοκληρωθούν όλα τα stages.

2.7 Πραγματική εκτέλεση

Κατά την πραγματική εκτέλεση το spark μπορεί να συμπεριφερθεί διαφορετικά από το αναμενόμενο. Ένα τέτοιο παράδειγμα είναι ότι το spark, για λόγους optimization, δεν περιμένει απαραίτητα για όλα τα tasks να ολοκληρωθούν και μπορεί να αγνοήσει μερικά αν δεν τα χρειάζεται. Επιπλέον, κάποια tasks μπορεί να πρέπει να δράσουν σε πολύ μεγάλα partitions ενός rdd ή το reduceByKey() να δημιουργεί πολύ μεγάλο shuffling λόγο της ύπαρξης πολλών unique keys. Σε αυτήν την περίπτωση, μπορεί οι executors του cluster να μην μπορούν να εκτελέσουν τα tasks με αποτέλεσμα την αποτυχία του app.



Σχήμα 2.3: Ροή εκτέλεσης

Κεφάλαιο 3

Έξυπνα Συμβόλαια (Smart Contracts)

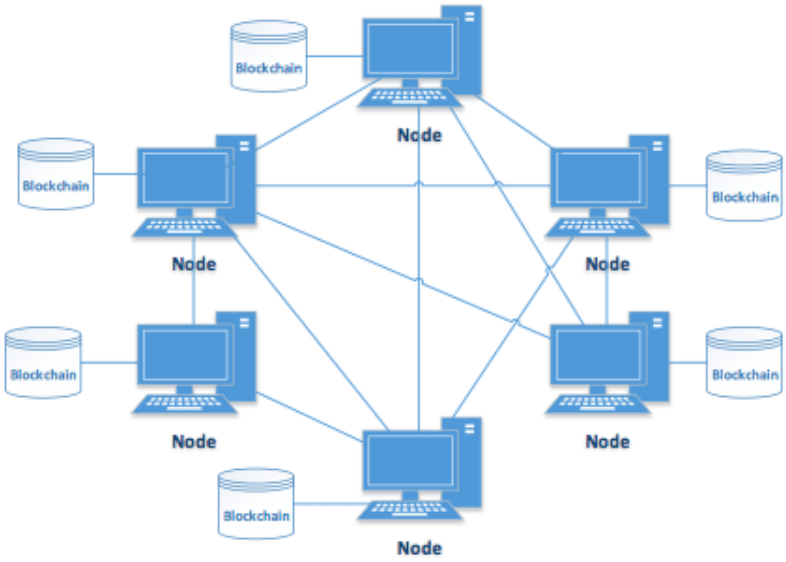
Σε αυτό το κεφάλαιο θα αναλυθεί η τεχνολογία των smart contract καθώς και του blockchain [3].

3.1 Εισαγωγή στο Blockchain

Ο όρος blockchain αναφέρεται σε μια κατανεμημένη δομή δεδομένων που αποτελείται από blocks τα οποία είναι συνδεδεμένα μεταξύ τους με hashes. Κάθε block περιέχει ένα hash που δείχνει στο προηγούμενο block, μια χρονοσφραγίδα και μια λίστα από συναλλαγές. Η χρονοσφραγίδα υπάρχει ώστε να αποδεικνύει ότι οι συναλλαγές δημιουργήθηκαν μετά το block. Αυτή η δομή δεδομένων διαχειρίζεται από ένα δίκτυο χρηστών(peer-to-peer), κάθε κόμβος του δικτύου έχει αποθηκευμένο ένα instance του blockchain και συμμορφώνεται με έναν αλγόριθμο συμφωνίας(consensus) για την εισαγωγή νέων κόμβων στο blockchain. Οι δύο γνωστότεροι αλγόριθμοι consensus είναι οι proof-of-work και proof-of-stake. Επιπλέον, το hash ενός block προκύπτει κατακερματιζοντας το block με κάποια συνάρτηση κατακερματισμού και χρησιμοποιείται ως σημείο αναφοράς του block. Τέλος, το blockchain και κάθε block από την στιγμή της εισαγωγής τους είναι immutable (αμετάβλητο).

Proof-of-work

Στο proof-of-work όλοι οι κόμβοι του δικτύου ανταγωνίζονται μεταξύ τους προκειμένου να βρουν έναν αριθμό που αν προστεθεί στα περιεχόμενα του block, τότε ο κατακερματισμός του block δημιουργεί ένα hash που τηρεί κάποιες προϋποθέσεις.



Σχήμα 3.1: Peer to peer δίκτυο blockchain

Για παράδειγμα, το hash ξεκινάει από πέντε μηδενικά. Ο πρώτος κόμβος που βρει τον αριθμό, κάνει broadcast το block και ανταμείβεται για την δουλειά του. Η διαδικασία αναζήτησης του αριθμού ονομάζεται mining.

Proof-of-stake

Σε αυτόν τον αλγόριθμο, οι κόμβοι που επιθυμούν να συμμετέχουν παραχωρούν ένα χρηματικό ποσό στο δίκτυο για όσο το επιθυμούν και έτσι αποκτούν το δικαίωμα να εγκυροποιούν blocks. Για κάθε block επιλέγονται τυχαία μερικοί κόμβοι, οι οποίοι κάνουν το validation και αν τα αποτελέσματα συμφωνούν, το block προστίθεται στο blockchain. Οι τυχεροί κόμβοι ανταμείβονται για την δουλειά τους.

Χρήστες του Blockchain

Προκειμένου κάποιος χρήστης να αποκτήσει χρήματα στο περιβάλλον κάποιο blockchain πρέπει να δημιουργήσει ένα wallet. Το wallet είναι ουσιαστικά ένα private και ένα public key που χρησιμοποιούνται για την υπογραφή συναλλαγών που θα σταλούν στο blockchain. Επιπλέον, το public key αποτελεί και την διεύθυνση του wallet του κάθε χρήστη. Με άλλα λόγια, κάθε φορά που μια συναλλαγή πραγματοποιείται, αυτό που συμβαίνει είναι ότι δημιουργείται μια εγγραφή στο τωρινό block που δείχνει ότι ένα wallet έστειλε σε κάποιο άλλο wallet ένα ποσό. Επομένως, το συνολικό χρηματικό ποσό κάθε wallet προκύπτει από το άθροισμα όλων των συναλλαγών που το wallet δέχεται χρήματα πλην τις συναλλαγές που το wallet στέλνει χρήματα.

Αποκέντρωση, Διαφάνεια και Ανωνυμία

Επειδή στα μεγάλα δίκτυα blockchain συμμετέχουν πάρα πολλοί κόμβοι από απλούς χρήστες, δεν υπάρχει κάποια αρχή που να ελέγχει ή να διέπει τις συναλλαγές. Επομένως, το blockchain προσφέρει αποκέντρωση. Επιπλέον, όλα τα records από συναλλαγές που έχουν γίνει validate είναι ανοιχτό προς όλους. Επομένως, υπάρχει διαφάνεια προς το ιστορικό του blockchain. Τέλος, ο κάθε χρήστης του blockchain είναι απλά ένα public key. Αυτό δίνει ανωνυμία στους χρήστες.

3.2 Ethereum

Το blockchain που θα χρησιμοποιηθεί στην παρούσα πτυχιακή εργασία είναι το Ethereum. Το Ethereum αποτελεί το δεύτερο μεγαλύτερο blockchain στον κόσμο, με πρώτο το bitcoin. Πρόκειται για ένα αποκεντρωμένο και open source blockchain. Μέχρι τις 15 Σεπτεμβρίου του 2022 χρησιμοποιούσε proof-of-work, ωστόσο πλέον χρησιμοποιεί proof-of-stake. Διαχειρίζεται από ένα πολύ μεγάλο peer-to-peer δίκτυο και έχει σαν νόμισμα του το ether. Τέλος, σε κάθε κόμβο του δικτύου Ethereum τρέχει μια turing complete εικονική μηχανή που ονομάζεται EVM(Ethereum Virtual Machine).

EVM(Ethereum Virtual Machine)

Η εικονική μηχανή του Ethereum είναι ένας υπολογιστής που τρέχει σε κάθε κόμβο του peer-to-peer δικτύου και μπορεί να πραγματοποιήσει τις βασικές λειτουργίες ενός υπολογιστή. Διαθέτει δική του μνήμη στην μορφή στοίβας και χρησιμοποιείται για να τρέξει προγράμματα υπολογιστή που ονομάζονται smart contracts στο περιβάλλον του Ethereum.

Γιατί Ethereum

Η επιλογή του ethereum γίνεται επειδή παρέχει την δυνατότητα εκτέλεσης smart contracts. Υπάρχουν και άλλα blockchain με αντίστοιχες δυνατότητες, όπως το cardano. Ωστόσο, επιλέγεται το Ethereum γιατί είναι πιο μπροστά στην ανάπτυξη του συγκριτικά με τον ανταγωνισμό και επίσης είναι πιο ευρέως διαδεδομένο.

3.3 Έξυπνα Συμβόλαια (smart contracts)

Τα Έξυπνα συμβόλαια, παρά την ιδιαίτερη ονομασία τους, αποτελούν προγράμματα υπολογιστή τα οποία τρέχουν στο περιβάλλον του Ethereum. Συγκεκριμένα, αποτελούν μεταφρασμένο κώδικα που γίνεται deploy στο blockchain και παρέχουν

κάποιες μεθόδους προς το υπόλοιπο δίκτυο. Το αναγνωριστικό ενός smart contract είναι μια διεύθυνση σαν αυτή των wallet. Ωστόσο, σε αντίθεση με τα wallets, τα transactions που αποστέλλονται στις διευθύνσεις των smart contract έχουν σκοπό να χρησιμοποιήσουν κάποια μέθοδο τους. Επισημαίνεται ότι ο κώδικας του smart contract τελικά εκτελείται από τους κόμβους που επιλέγονται να κάνουν validate το block στο οποίο ανήκει η συναλλαγή που χρησιμοποιεί τις μεθόδους του. Επομένως, η εκτέλεση του smart contract υπόκειται στον επιλεγμένο αλγόριθμο consensus. Κλείνοντας, θα μπορούσε να γίνει η αντιστοιχία μεταξύ ενός smart contract και ενός web server που παρέχει μερικά api. Το url του server θα αντιστοιχεί στην διεύθυνση του smart contract και τα api του server θα αντιστοιχούν στις μεθόδους των smart contract.

Γλώσσες Προγραμματισμού

Υπάρχουν διάφορες γλώσσες προγραμματισμού που μπορούν διαθέτουν compiler που μεταγράζει τον κώδικα του smart contract σε εκτελέσιμο έτοιμο να τρέξει στο blockchain. Μερικές τ΄τετοιες γλώσσες είναι LLL, Serpent, Vyper και Bamboo. Ωστόσο, στα πλαίσια αυτής της διπλωματικής θα χρησιμοποιηθεί η Solidity, η οποία αποτελεί την πιο διαδεδομένη γλώσσα προγραμματισμού για smart contracts. Η Solidity είναι μια αντικειμενοστραφής, υψηλού επιπέδου γλώσσα προγραμματισμού που μοιάζει με την C++ και την java.

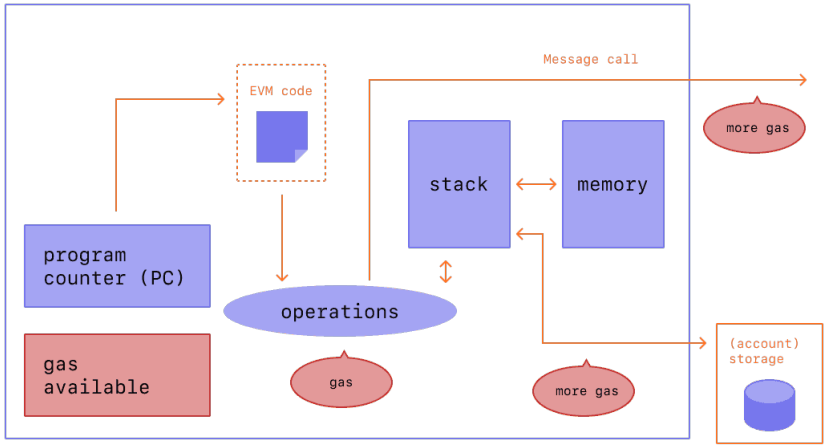
Αποθηκευτικός Χώρος

Τα smart contract διαθέτουν δύο τύπους μνήμης. Ο πρώτος τύπος ονομάζεται memory και χρησιμοποιείται για την αποθήκευση δεδομένων προσωρινά κατά την διάρκεια εκτέλεσης μιας μεθόδου. Με άλλα λόγια, ότι αποθηκεύεται στο memory, αποθηκεύεται στην μνήμη του EVM μέχρι το πέρας του τωρινού call και μετά διαγράφεται. Έτσι, επόμενα calls δεν θα έχουν πρόσβαση σε δεδομένα που αποθηκεύονται στο memory από προηγούμενα calls.. Ο δεύτερος τύπος ονομάζεται storage και ουσιαστικά σημαίνει την μόνιμη αποθήκευση των δεδομένων στο blockchain. Δεδομένα που αποθηκεύονται στο storage παραμένουν για πάντα διαθέσιμα όσο ζει το smart contract. Αξίζει να σημειωθεί, ότι δεν υπάρχει κάποια ανάγκη κλειδώματος όταν γίνεται εγγραφή στο storage επειδή δεν υπάρχει η έννοια της ταυτόχρονης εγγραφής. Αντίθετα, η σειρά με την οποία πραγματοποιούνται η εγγραφές έχει να κάνει με την σειρά που εισέρχονται στα blocks.

Κόστος σε Gas

Η αποστολή ενός transaction στο blockchain κοστίζει ένα είδος fee που ονομάζεται gas. Το gas υπολογίζεται για κάθε transaction ανάλογα με το πόση ώρα θα απασχολήσει το blockchain. Για παράδειγμα, μια απλή συναλλαγή μεταξύ δύο wallets θα κοστίσει λιγότερο από ένα κάλεσμα μεθόδου smart contract που θα χρειαστεί υπολογιστικές πράξεις. Σημαντικός δείκτης του πόσο καλό είναι ένα smart contract

αποτελεί το πόσο λίγο gas καταναλώνει το κάλεσμα των μεθόδων του. Επισημαίνεται, όπως είναι αναμενόμενο, ότι η εγγραφή στο memory είναι φθηνότερη από την εγγραφή στο storage. Τέλος, για τον υπολογισμό του κόστους μια συναλλαγής σε ether, το gas έχει ένα price σε ether και πραγματοποιείται πολλαπλασιασμός.



Σχήμα 3.2: EVM και gas

Call vs Transact

Υπάρχουν δύο τρόποι για το κάλεσμα μιας μεθόδου ενός smart contract. Η πρώτη είναι μέσω ενός transaction. Με αυτόν τον τρόπο μπορεί να πραγματοποιηθεί εγγραφή τόσο στο memory όσο και στο storage. Το αρνητικό είναι ότι πάντα πρέπει να πληρώνεται το gas. Ωστόσο, υπάρχει και ένας δεύτερος τρόπος που ονομάζεται call. Αυτός ο τρόπος χρησιμοποιείται μόνο για την επιστροφή τιμών στον χρήστη και δεν μπορεί να πραγματοποιηθεί εγγραφή στο storage. Συγκεκριμένα, δεν απασχολεί ολόκληρο το δίκτυο αλλά μόνο τον κόμβο με τον οποίο συνδέεται ο χρήστης και αποτελεί μια απλή ανάγνωση του blockchain. Το θετικό είναι ότι ενώ υπολογίζεται το gas δεν χρειάζεται να πληρωθεί. Παρόλα αυτά, μια κλήση με call μπορεί να μην πραγματοποιηθεί αν το gas που καταναλώνει ξεπεράσει το επιτρεπτό όριο.

Τύποι και Δομές Δεδομένων στο Solidity

Η γλώσσα προγραμματισμού Solidity διαθέτει πολλούς από του κλασσικούς τύπους δεδομένων όπως boolean, int, struct και array. Επίσης, διαθέτει μερικούς τύπους που βοηθούν στην λειτουργικότητα των smart contracts όπως ο τύπος address και ο τύπος Ether units. Επιπλέον, υπάρχει ο τύπος Mapping ο οποίο αποτελεί ένα hash map που αντιστοιχεί ένα κλειδί σε μία τιμή. Στα πλαίσια της παρούσας πτυχιακής θα χρησιμοποιηθούν κυρίως οι δομές struct και Mapping. Πρέπει να τονιστεί ότι στο solidity τόσο οι μεταβλητές όσο και οι δομές δεδομένων δεν απαιτούν αρχικοποίηση. Συγκεκριμένα, ανάλογα με τον τύπο της, η κάθε μεταβλητής έχει μια

default τιμή. Για παράδειγμα, οι μεταβλητές bool αρχικοποιούνται πάντα ως false, ενώ η μεταβλητές int αρχικοποιούνται στο μηδέν. Κάτι αντίστοιχο συμβαίνει και στην δομή mapping. Με άλλα λόγια, μια δομή Mapping μόλις δηλωθεί περιέχει όλα τα κλειδιά αρχικοποιημένα στην default τιμή τους. Για παράδειγμα, αν έχει δηλωθεί ένα Mapping με keys τύπου int και τιμές τύπου int, τότε ένα τυχαίο key, πχ το 5 θα αντιστοιχεί σε τιμή 0 μέχρι κάποιο call του smart contract αλλάξει την τιμή του.

Πλεονέκτημα των Smart Contract

Όπως αναφέρθηκε παραπάνω, τα έξυπνα συμβόλαια λειτουργούν σαν προγράμματα υπολογιστή. Ωστόσο, η κύρια διαφορά τους συγκριτικά με ένα παραδοσιακό πρόγραμμα που τρέχει σε ένα μηχάνημα είναι ότι οι υπολογισμοί που συμβαίνουν στο Smart Contract πρέπει να επαληθευτούν από τον αλγόριθμο consensus του Blockchain. Με άλλα λόγια, όσο μεγαλύτερο είναι το peer to peer δίκτυο, τόσο πιο δύσκολο είναι κάποιος κακόβουλος να μπορέσει να επηρεάσει την εκτέλεση των μεθόδων ενός smart contract. Ουσιαστικά, το δίκτυο του Ethereum είναι αρκετά μεγάλο που καθιστά την κακόβουλη εκμετάλλευση πρακτικά αδύνατη. Επομένως, ένα smart contract μπορεί να λειτουργήσει ως μια αξιόπιστη αρχή, την οποία εμπιστεύονται όλα τα μέλη ενός Apache Spark cluster.

Κεφάλαιο 4

Αξιοπιστία στο Apache Spark, κίνδυνοι και λύσεις

Σε αυτό το κεφάλαιο θα εξεταστούν παράγοντες που μπορεί να αλλοιώσουν τα αποτελέσματα του Apache Spark, μειώνοντας έτσι την αξιοπιστία του εργαλείου. Επιπλέον, θα γίνει προσπάθεια εύρεσης λύσης για κάθε πρόβλημα που εντοπίζεται.

4.1 Ανοιχτοί Clusters

Όπως αναφέρθηκε και στην εισαγωγή, υπάρχουν δύο είδη cluster. Καταρχάς, υπάρχουν οι clusters που ελέγχονται από κάποια αρχή και επιτρέπουν την συμμετοχή σε ελεγχόμενα μηχανήματα. Αυτοί οι clusters προσφέρουν ένα έμπιστο περιβάλλον κατανεμημένης επεξεργασίας μιας και υπάρχει εγγύηση ότι όλοι οι κόμβοι είναι τίμιοι και εκτελούν σωστά τις εργασίας που τους αναθέτονται. Ωστόσο, τέτοιοι clusters αντιμετωπίζουν προβλήματα με την κλιμακωσιμότητα(scalability). Επομένως, με την συνεχή αύξηση του όγκου δεδομένων προς επεξεργασία, έχουν εμφανιστεί ανοιχτοί clusters που μπορεί να συμμετέχει όποιος το επιθυμεί, λύνοντας έτσι το πρόβλημα του scalability. Παρόλα αυτά, αφήνοντας οποιονδήποτε να συμμετέχει, οι ανοιχτοί clusters παρουσιάζουν προβλήματα μειωμένης αξιοπιστίας. Συγκεκριμένα, σε ανοιχτούς clusters συμμετέχουν κόμβοι workers είτε εθελοντικά είτε για κάποια αμοιβή. Στα πλαίσια της παρούσας πτυχιακής εργασίας θα μελετηθεί κυρίως η συμμετοχή σε έναν cluster που τρέχει Apache Spark για κάποια αμοιβή. Υπάρχουν αρκετά παραδείγματα αμοιβών όπως η χρηματική ή το δικαίωμα εκτέλεσης εργασιών στον cluster. Επομένως, ο λογικός στόχος ενός κόμβου είναι η μεγιστοποίηση της αμοιβής για την λιγότερη δυνατή παροχή πόρων. Αντίστοιχα, οι χρήστες που υποβάλουν εργασίες στον cluster πρέπει επίσης να πληρώσουν κάποιο αντίτιμο. Παραδείγματα αντίτιμου

μπορεί να είναι ένα χρηματικό ποσό ή η εξαργύρωση του δικαιώματος χρήσης του cluster που κερδήθηκε ως αμοιβή για την προσφορά πόρων. Επομένως, ο λογικός στόχος ενός χρήστη είναι η ορθή εκτέλεσης των εργασιών του. Κατά συνέπεια ένας χρήστης επιθυμεί τον έλεγχο των αποτελεσμάτων και σε περίπτωση λάθους την επιστροφή του αντιτίμου που πλήρωσε. Επιπλέον, ο master του cluster αποτελεί έναν κόμβο και άρα έχει τους ίδιους στόχους με τους υπόλοιπους. Στην συνέχεια, θα μελετηθούν οι κίνδυνοι που μπορεί να βλάψουν την αξιοπιστία του Apache Spark σε ανοιχτό cluster και θα προταθούν λύσεις. Τέλος, αξίζει να σημειωθεί ότι οι κίνδυνοι που θα παρουσιαστούν παρακάτω υφίστανται και σε clusters με εθελοντική συμμετοχή. Ωστόσο, σε αυτήν την περίπτωση κάποιες συμπεριφορές δεν έχουν λογικό κίνητρο. Επομένως, οι κίνδυνοι ενός cluster με εθελοντική συμμετοχή αποτελούν υποσύνολο των κινδύνων ενός cluster με συμμετοχή για αμοιβή.

4.2 Κίνδυνοι και Λύσεις

4.2.1 Λανθασμένη εκτέλεση Task

Η πιο απλή και συνηθισμένη αιτία αλλοίωσης των αποτελεσμάτων ενός spark app είναι η λάθος εκτέλεση ενός task από κάποιον worker. Ένα τέτοιο λάθος μπορεί να μην είναι εσκεμμένο και να προκύψει τυχαία από κάποιον τίμιο worker. Ωστόσο, μπορεί να γίνει ηθελημένα από κάποιον worker που προσπαθεί να λάβει την αμοιβή για την εκτέλεση των εργασιών που του ανατέθηκαν χωρίς να δαπανήσει την υπολογιστική ισχύ που χρειάζεται. Και στις δύο παραπάνω περιπτώσεις μειώνεται η αξιοπιστία του Apache Spark. Για να επιλυθεί αυτός ο κίνδυνος χρησιμοποιείται η αντιγραφή(replication) του κάθε task σε παραπάνω από ένα τα οποία τρέχουν από διαφορετικούς workers με σκοπό την σύγκριση των αποτελεσμάτων και την εύρεση λαθών. Σε επόμενο κεφάλαιο θα αναλυθούν οι παράγοντες που επηρεάζουν την επιλογή αριθμού αντιγράφων.

4.2.2 Εκτέλεση Spark Driver σε μη έμπιστο κόμβο

Όπως αναλύθηκε στο Κεφάλαιο 2, το σημαντικότερο μέρος μιας εφαρμογής spark είναι ο spark driver, ο οποίος είναι υπεύθυνος για την δημιουργία του πλάνου εκτέλεσης και την αποστολή των tasks στους executors. Ουσιαστικά, ο spark driver καθορίζει αν η εφαρμογή που θα εκτελεστεί είναι η εφαρμογή που υπέβαλε ο χρήστης. Επομένως, αποτελεί μεγάλο κίνδυνο η εκτέλεση του driver σε μη τίμιο worker. Με άλλα λόγια, ένας μη τίμιος worker μπορεί είτε να μην δημιουργήσει σωστά το πλάνο εκτέλεσης για να γλυτώσει πόρους είτε να εκμεταλλευτεί την ευκαιρία και να τρέξει μια δική του εφαρμογή ενώ πληρώνει το αντίτιμο ο πελάτης. Ευτυχώς, η επίλυση αυ-

τού του προβλήματος είναι αρκετά εύκολη. Το spark δίνει το δικαίωμα να επιλεγεί η τοποθεσία που θα τρέξει ο spark driver κατά την υποβολή της εφαρμογής. Επομένως, κατά το spark-submit πρέπει να επιλεχθεί σαν deploy-mode το client προκειμένου να εκτελεστεί ο spark driver στο μηχάνημα του πελάτη. Με αυτόν τον τρόπο υπάρχει βεβαιότητα ότι το πλάνο εκτέλεσης της εφαρμογής θα είναι σωστό.

4.2.3 Προσπάθεια Client να μην πληρώσει

Τρέχοντας τον spark driver στο μηχάνημα του χρήστη, καθιστά τον πελάτη τον μοναδικό που έχει πρόσβαση στα αποτελέσματα των tasks και κατά συνέπεια και των αντιγραμμένων tasks. Επομένως, μόνο ο πελάτης μπορεί να πραγματοποιήσει τον έλεγχο και να ισχυριστεί αν η εφαρμογή εκτελέστηκε σωστά ή όχι. Αυτό μπορεί να προκαλέσει πρόβλημα αν ο πελάτης θελήσει να πει ψέματα ότι τα αποτελέσματα ήταν λανθασμένα και ζητήσει πίσω το αντίτιμο που πλήρωσε. Για την επίλυση αυτού του κινδύνου γίνεται η χρήση ενός έξυπνου συμβολαίου στο οποίο στέλνονται από τους executors και αποθηκεύονται τα αποτελέσματα των tasks σε μορφή hash(result hash) προκειμένου να έχουν όλοι οι εμπλεκόμενοι πρόσβαση. Επισημαίνεται ότι το έξυπνο συμβόλαιο προσφέρει αποκεντρωμένη και αμερόληπτη αποθήκευση των δεδομένων.

Επιπλέον, μιας και ο πελάτης τρέχει το spark driver είναι υπεύθυνος για την αντιγραφή των tasks. Επομένως, μπορεί να προκαλέσει εσκεμμένα σφάλματα εισάγωντας διαφορετικά tasks σαν ίδια. Με αυτόν τον τρόπο, tasks που θα έπρεπε να έχουν το ίδιο αποτέλεσμα εμφανίζουν λάθος και ο client μπορεί να μην πληρώσει. Για να επιλυθεί αυτό το πρόβλημα ο spark driver για κάθε task δημιουργεί μια ψηφιακή υπογραφή που περιέχει το task index, το stageId, το appId και ένα hash που προκύπτει από το binary code του κώδικα του task. Στην συνέχεια, στέλνει αυτή την ψηφιακή υπογραφή στον executor μαζί με το task. Ο executor με την σειρά του μαζί με το result hash στέλνει στο έξυπνο συμβόλαιο και την ψηφιακή υπογραφή. Αυτή η ψηφιακή υπογραφή εξασφαλίζει δύο πράγματα. Πρώτον, ότι ο driver έστειλε τα σωστά task στους executors. Δεύτερον, ότι ο executor που στέλνει το result hash ήταν αυτός που του είχε ανατεθεί το task. Ακόμα, ο πελάτης αποθηκεύει την ψηφιακή υπογραφή για να μπορέσει αργότερα να αποδείξει ότι υλοποίησε ορθά την διαδικασία αντιγραφής των tasks.

Επίσης, ο πελάτης είναι ο μοναδικός που γνωρίζει πόσα και ποια tasks ολοκληρώθηκαν και έστειλαν πίσω αποτέλεσμα. Έτσι, μπορεί να ισχυριστεί ότι κάποιο task που δεν ολοκληρώθηκε ποτέ έστειλε λάθος αποτέλεσμα ή ότι ο αντίστοιχος executor δεν έστειλε στο έξυπνο συμβόλαιο το result hash. Αυτός ο κίνδυνος μπορεί να αντιμετωπιστεί αν ο κάθε executor αντί για το result hash στείλει στο έξυπνο συμβόλαιο μια ψηφιακή υπογραφή που περιέχει το task index, το stageId, το appId και το result hash. Την ίδια ψηφιακή υπογραφή θα στείλει και στον spark driver, ο οποίος με την σειρά του θα αποθηκεύσει.

Τελικά, ο πελάτης θα έχει αποθηκευμένες για κάθε task δύο ψηφιακές υπογραφές, την δικιά του με το hash του κώδικα εκτέλεσης τους task και του executor με το result hash. Άρα, προκειμένου να αποδείξει ότι τα tasks εκτελέστηκαν, ολοκληρώθηκαν και να εντοπίσει κάποιο λάθος θα πρέπει μετά την ολοκλήρωση της εφαρμογής να στείλει στο έξυπνο συμβόλαιο τις δύο ψηφιακές υπογραφές. Έχοντας όλες τις ψηφιακές υπογραφές, το έξυπνο συμβόλαιο μπορεί να πραγματοποιήσει τους απαραίτητους ελέγχους και να εντοπίσει τυχόν λάθη.

4.2.4 Συνεργασία Spark Master με Spark Workers

Σε έναν ανοιχτό cluster υπάρχει η περίπτωση συνεργασίας μερικών workers με στόχο την μεγιστοποίηση του κέρδους. Στην περίπτωση του apache spark με αντιγραφή tasks, η συνεργασία κόμβων μπορεί να δημιουργήσει πρόβλημα στην αξιοπιστία της διαδικασίας. Με άλλα λόγια, αν μια πλειάδα αντιγραμμένων tasks ανατεθεί σε workers που συνεργάζονται μπορούν να μην πραγματοποιήσουν την εργασία και να συμφωνήσουν στο αποτέλεσμα. Ωστόσο, η δρομολόγηση των tasks γίνεται από τον client και επομένως η αλλοίωση των αποτελεσμάτων βασίζεται στην τύχη και δεν είναι αποδοτική. Στην περίπτωση, όμως, που στην συνεργασία συμμετέχει και ο spark master μπορεί να δημιουργηθούν περισσότερα προβλήματα. Επισημαίνεται ότι ο spark driver επιλέγει ποιο tasks θα εκτελεστεί από ποιον διαθέσιμο executor Άρα, δεν μπορεί ο master να υποχρεώσει τον driver να αναθέσει συγκεκριμένα tasks σε συγκεκριμένους executors. Παρόλα αυτά, ο master μπορεί να εμφανίζει συχνότερα διαθέσιμους τους executors των workers με τους οποίους συνεργάζεται. Με αυτόν τον τρόπο, μπορεί να αυξήσει την πιθανότητα μια πλειάδα αντιγραμμένα tasks να ανατεθεί στους workers που συνεργάζονται, μειώνοντας έτσι την αξιοπιστία του apache spark. Για να επιλυθεί αυτό το πρόβλημα, το έξυπνο συμβόλαιο μπορεί να υπολογίζει πόσα tasks ανατέθηκαν σε κάθε worker και να εντοπίσει workers που χρησιμοποιούνται υπερβολικά. Τονίζεται, ότι δεν έχουν όλοι οι workers ίδια υπολογιστική ισχύ και άρα είναι φυσιολογικό ένας worker να χρησιμοποιείτε περισσότερο από κάποιον άλλο. Επομένως, για να έχει νόημα η παραπάνω μέτρηση πρέπει να γίνουν κάποιες παραδοχές για την διαφορά στην υπολογιστική ισχύ του ισχυρότερου και του πιο αδύναμου worker. Στα πλαίσια αυτής της πτυχιακής, γίνεται η παραδοχή ότι ο ισχυρότερος worker διαθέτει διπλάσια υπολογιστική ισχύ συγκριτικά με τον πιο αδύναμο worker του cluster.

Κεφάλαιο 5

Επέκταση του Apache Spark

Σε αυτό το κεφάλαιο θα παρουσιαστεί η αρχιτεκτονική της επέκτασης του apache spark και ο τρόπος λειτουργίας της. Επισημαίνεται ότι προκειμένου να πραγματοποιηθεί η επικοινωνία με το smart contract τόσο οι κόμβοι του cluster όσο και οι χρήστες πρέπει να διαθέτουν ένα wallet. Το wallet ουσιαστικά αποτελείται από έναν κρυφό και έναν φανερό κωδικό. Επομένως, οι ψηφιακές υπογραφές που αναφέρθηκαν στο προηγούμενο κεφάλαιο μπορούν δημιουργηθούν με το ζευγάρι κωδικών των wallets.

5.1 Overview

Όπως έχει ήδη αναφερθεί, η επέκταση του Spark που υλοποιήθηκε σε αυτήν την εργασία χρησιμοποιεί την μέθοδο του replication. Με άλλα λόγια, στον spark driver πραγματοποιείται η εισαγωγή του κάθε task παραπάνω από μια φορά στο taskset. Με αυτόν τον τρόπο, το κάθε task εκτελείται σε διαφορετικούς executors παραπάνω από μία φορα και συνεπώς μπορεί να γίνει η σύγκριση των αποτελεσμάτων των αντιτύπων του ίδιου task και να βρεθούν πιθανά λάθη. Η διαδικασία του verification γίνεται στο Smart Contract. Συγκεκριμένα, η επικοινωνία με το smart contract γίνεται σε δύο φάσεις. Κατά την διάρκεια της εκτέλεσης της εφαρμογής οι executors στέλνουν στο smart contract δύο hashes, το TaskHash που αποτελεί το hash του εκτελέσιμου κώδικα του task ψηφιακά υπογεγραμμένο από τον client και απεσταλμένο από τον executor, και το resultHash που είναι το hash του αποτελέσματος του task υπογεγραμμένο από τον executor. Ταυτόχρονα, μόλις ολοκληρωθεί ένα task και σταλθεί το αποτέλεσμα στον client, ο spark driver αναλαμβάνει να αποθηκεύσει το ψηφιακά υπογεγραμμένο resultHash που δέχτηκε από τον executor και το TaskHash που έχει υπογράψει ο ίδιος, σε ένα αρχείο επιβεβαίωσης. Τελικά, μετά την ολοκλήρωση της εφαρμογής, ο driver μπορεί να στείλει τα hashes από το αρχείο επιβεβαίωσης στο

smart contract προκειμένου να γίνει το verification. Για να πραγματοποιηθούν τα παραπάνω αναπτύχθηκε ένα smart contract και πραγματοποιήθηκαν κάποιες παρεμβάσεις στο Apache Spark. Τόσο η λειτουργία του smart contract όσο και η αλλαγές στο Apache Spark θα αναλυθούν παρακάτω.

5.2 Smart contract

Για την επιβεβαίωση της επεξεργασίας του Apache Spark θα χρησιμοποιηθεί ένα έξυπνο συμβόλαιο. Σκοπός του συμβολαίου αυτού είναι η αποκεντρωμένη αποθήκευση των αποτελεσμάτων των tasks και ο έλεγχος της ορθότητας τους. Για να το πετύχει αυτό το Smart Contract χρησιμοποιεί τις εξής δομές δεδομένων:

- taskData: Η συγκεκριμένη δομή αποτελεί ένα struct το οποίο χρησιμοποιείται για να αποθηκεύσει τα δεδομένα του κάθε task. Ουσιαστικά, καθε struct taskData περιμένει σαν δεδομένα το driverTaskHash, το execTaskHash, το resultHash και δύο λογικές μεταβλητές, την exec και την posted. Το driverTaskHash αποτελεί το hash του εκτελέσιμου κώδικα του task ψηφιακά υπογεγραμμένο από τον driver και απεσταλμένο στο smart contract επίσης από τον driver. Το execTaskHash είναι το ίδιο ψηφιακά υπογεγραμμένο hash αλλά απεσταλμένο από τον executor. Το resultHash είναι το hash του αποτελέσματος του task υπογεγραμμένο από τον executor και απεσταλμένο στο smart contract. Τέλος, οι μεταβλητές exec και posted γίνονται true μόλις ο executor και ο driver αντίστοιχα στείλουν τα δεδομένα τους.
- taskSet: Η δομή taskSet αποτελεί ένα mapping που αντιστοιχεί ακεραίους task index σε structs taskData.
- stages: Η δομή stages αποτελεί ένα struct που περιέχει ένα mapping το οποίο αντιστοιχεί ακεραίους stageId σε mappings taskSet. Επιπλέον, περιέχει μια ακέραια μεταβλητή status που εκφράζει την κατάσταση της κάθε εφαρμογής.
- app: Η δομή app αποτελεί ένα mapping που αντιστοιχεί strings appId σε structs stages. Επομένως για κάθε task που έχει σταλεί στο smart contract μπορεί να γίνει ανάκτηση των πληροφοριών του αν γνωρίζουμε τον appid, το stageid και το task index με τον εξής τρόπο app[appId].stages[stageId].taskSet[tid].taskdata.
- idToHostname: Αυτή η δομή είναι ένα mapping που αντιστοιχεί ένα worker id σε ένα hostname κάποιου worker.
- hostnameToId: Αυτή η δομή είναι ένα mapping που αντιστοιχεί ένα hostname κάποιου worker σε ένα worker id. Ουσιαστικά συμπληρώνει την δομή idToHostname και καθιστά εύκολη την εύρεση id από hostname και hostname από id.

- idToCount: Αυτή η δομή αποτελεί ένα mapping που αντιστοιχεί κάθε worker id στο αριθμό των tasks που έχει ολοκληρώσει.
- totalWorkers: Μια ακέραια μεταβλητή που μετράει τον συνολικό αριθμό των workers.
- totaltasks: Μια ακέραια μεταβλητή που μετράει τον συνολικό αριθμό των tasks.

Έχοντας παρουσιάσει τις δομές δεδομένων που χρησιμοποιεί το smart contract, ακολουθούν οι μέθοδοι:

- verifyPair: Αυτή η μέθοδος δέχεται σαν όρισμα δύο task ids, ένα stageid και ένα appid και πραγματοποιεί τον έλεγχο των αποτελεσμάτων. Η verifyPair επιστρέφει την τιμή μηδέν αν τα αποτελέσματα είναι σωστά. Ωστόσο, σε περίπτωση που δεν είναι σωστά επιστρέφει:
  - -2 αν η μεταβλητή posted κάποιου task είναι false που σημαίνει ότι ο spark driver δεν έχει στείλει τα δεδομένα του ακόμα.
  - -1 αν η μεταβλητή exec κάποιου task είναι false που σημαίνει ότι ένας από τους δύο executors δεν έστειλαν το result hash.
  - 1 αν τα result hash διαφέρουν αλλά οι μεταβλητές driverTaskHash και είναι σωστές. Σε αυτή την περίπτωση υπάρχει λαθος στην εκτέλεση των tasks
  - 2 αν τα resulthash διαφέρουν, τα driverTaskHash και execTaskHash είναι ίδια για το κάθε task ωστόσο τα δύο execTaskHash διαφέρουν μεταξύ του. Σε αυτήν την περίπτωση ο spark driver έστειλε διαφορετικά tasks ενώ έπρεπε να είχε στείλει ίδια. Επομένως, το λάθος αποτέλεσμα οφείλεται στον client.
  - 3 αν τα resulthash διαφέρουν και τα driverTaskHash και execTaskHash επίσης διαφέρουν για κάποιο από τα δύο tasks. Σε αυτήν την περίπτωση πάλι οφείλεται στον client το λάθος επειδή έστειλε διαφορετικό hash του εκτελέσιμου κώδικα του task στον executor και διαφορετικό στο smart contract. Επισημαίνεται ότι η τα driverTaskHash και execTaskHash είναι ψηφιακά υπογεγραμμένα από τον driver και περιέχουν το appId, το task index και το stageId επομένως ο executor δεν μπορεί να στείλει ψεύτικο execTaskHash.

  Τονίζεται ότι το verifyPair λειτουργεί για ζευγάρια ίδιων tasks. Ωστόσο, μπορεί να επεκταθεί και για μεγαλύτερο αριθμό αντιγραφής.
- addResultfromExec: Αυτή η μέθοδος καλείται από τον executor προκειμένου να στείλει το execTaskHash και το resulthash στο smart contract. Η ίδια μέθοδος κάνει και το exec του κάθε task true. Επιπλέον, σε αυτήν την μέθοδο

αυξάνεται και το idToCount για τον worker που έστειλε το task μαζί με το totaltasks.

- updatePostedTaskPair: Αυτή η μέθοδος καλείται, μετά το πέρας της εφαρμογής, από τον driver προκειμένου να στείλει για κάθε ζευγάρι από tasks που ολοκληρώθηκαν το driverTaskHash στο smart contract. Ταυτόχρονα κάνει την μεταβλητή posted των tasks true και καλεί το verifyPair. Σε περίπτωση που το verifyPair επιστρέψει τιμή διαφορετική του μηδέν αλλάζει την τιμη του status του struct stages και χαρακτηρίζει την εφαρμογή ως εσφαλμένη. Επισημαίνεται, ότι προκειμένου ο driver να καλέσει επιτυχημένα την updatePostedTaskPair πρέπει να στείλει στο smart contract και το ψηφιακά υπογεγραμμένο resultHash από τον executor προκειμένου να αποδείξει ότι το task ολοκληρώθηκε. Και εδώ τονίζεται ότι το updatePostedTaskPair λειτουργεί για ζευγάρια ίδιων tasks. Ωστόσο, μπορεί να επεκταθεί και για μεγαλύτερο αριθμό αντιγραφής.

- checkData: Αυτή η μέθοδος επιστρέφει τα δεδομένα από το taskData για κάποιο task.

- returnAppStatus: Αυτή η μέθοδος επιστρέφει το status κάποιου app από το struct stages.

- returnTotalWorkers: Αυτή η μέθοδος επιστρέφει τον συνολικό αριθμό των workers που έχουν στείλει κάποιο task στο smart contract.

- returnWorkerUsage: Αυτή η μέθοδος επιστρέφει το ποσοστό χρησιμοποίησης που προκύπτει από το idToCount[worker id] προς το totaltasks για κάποιο worker id.

5.3 Παρεμβάσεις στο Apache Spark

5.3.1 Παρεμβάσεις στον Spark Driver

Παρακάτω θα αναλυθούν οι αλλαγές που πρέπει να γίνουν στον Spark Driver.

sparkContext

Η πρώτη αλλαγή του driver γίνεται στο sparkContext και αφορά την δημιουργία ενός αρχείου για την αποθήκευση των δεδομένων που θα χρειαστεί ο client για την επιβεβαίωση της επεξεργασίας. Ουσιαστικά, το sparkContext δημιουργεί ένα αρχείο με όνομο το appid στο οποίο θα αποθηκευτούν στην συνέχεια οι ψηφιακές υπογραφές για τα tasks που ολοκληρώνονται. Αυτό το αρχείο θα ονομάζεται αρχείο επιβεβαίωσης.

Taskset manager

Στον taskset manager γίνεται μια τροποποίηση που αφορά την δημιουργία της ψηφιακής υπογραφής που περιέχει τα task index, stage id, app id και το hash του κώδικα εκτέλεσης του task. Στην συνέχεια, αποθηκεύεται αυτή η υπογραφή σε μια απλή δομή map προκειμένου αργότερα να έχει πρόσβαση ο DAGScheduler. Τελικά, στέλνεται μαζί με το task στον executor που επιλέχθηκε.

DAGScheduler

Η σημαντικότερες αλλαγές συμβαίνουν στο DAGScheduler. Καταρχάς, υλοποιείται η πολλαπλή εισαγωγή των ίδιων tasks σε κάθε taskset. Με άλλα λόγια, κατά την διάρκεια υπολογισμού των tasks ανά stage το κάθε task εισάγεται στο taskset παραπάνω από μια φορά με σκοπό την πολλαπλή εκτέλεση του από τους workers και την σύγκριση των αποτελεσμάτων στο τέλος της εφαρμογής. Στα πλαίσια αυτής της υλοποίησης το κάθε task εισάγεται δύο φορές. Ωστόσο, παρακάτω θα μελετηθεί και η εισαγωγή των task παραπάνω από δύο φορές. Επιπλέον, πραγματοποιείται και μια ακόμα παρέμβαση. Ουσιαστικά, πριν σταλούν τα αποτελέσματα στις συναρτήσεις που τα διαχειρίζονται ο DAGScheduler περιμένει να ολοκληρωθεί ολόκληρη η πλειάδα ίδιων tasks. Στην συνέχεια, για κάθε πλειάδα αποθηκεύει στο αρχείο επιβεβαίωσης τα task index, stage id, app id, την ψηφιακή υπογραφή που δημιούργησε ο Taskset manager, και έστειλε στον executor, και την ψηφιακή υπογραφή του hash του αποτελέσματος που έλαβε από τον executor.

TaskScheduler

Στον κώδικα του Task Scheduler πραγματοποιήθηκαν αλλαγές σε κάποιες ιδιωτικές συναρτήσεις προκειμένου να μην επιτρέπεται τα δύο αντίτυπα που ίδιου Task να ανατεθούν σε executors του ίδιου Worker. Επομένως, αυτός ο έλεγχος προστίθεται στις παραμέτρους που θα λάβει υπόψη ο taskScheduler προκειμένου να επιλέξει σε ποιον executor θα αναθέσει το κάθε task.

5.3.2 Παρεμβάσεις στον Spark Worker

Executor

Στον κώδικα του spark worker η μοναδική παρέμβαση αφορά τον κώδικα του executor. Συγκεκριμένα, ο executor μετά την ολοκλήρωση της εκτέλεσης του κάθε task δημιουργεί ένα hash από το αποτέλεσμα και το υπογράφει ψηφιακά μαζί με τα task index, stage id, app id. Στην συνέχεια, καλεί την μέθοδο addResultfromExec του smart contract προκειμένου να στείλει το αποτέλεσμα. Επισημαίνεται ότι η μέθοδος addResultfromExec καλείται με transact, επομένως τα αποτελέσματα αποθηκεύονται στο blockchain. Επιπλέον, μαζί με τα αποτελέσματα του κάθε task στέλνει στον

spark driver και την ψηφιακή υπογραφή.

5.3.3 Παρεμβάσεις στον Spark Master

Δεν πραγματοποιείται κάποια παρέμβαση στον Spark Master.

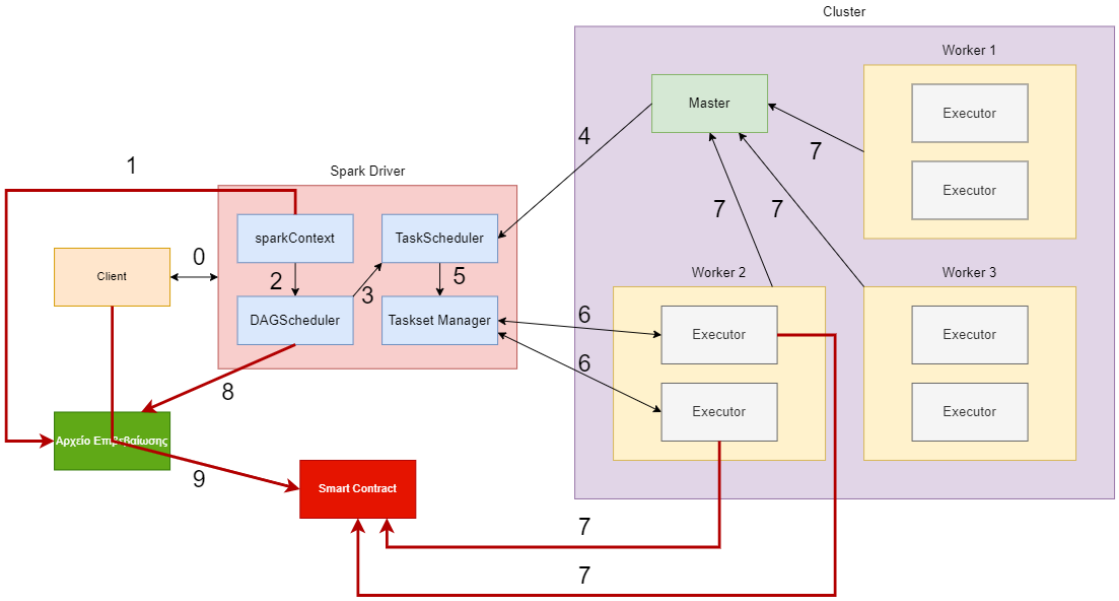
5.4 Διαδικασία Επιβεβαίωσης

Η διαδικασία επιβεβαίωσης χωρίζεται σε τρεις φάσεις και αρχικοποιείται από τον client προκειμένου να ελέγξει αν τα αποτελέσματα της εφαρμογής είναι σωστά. Καταρχάς, στην πρώτη φάση, ο client για κάθε task που αποθήκευσε στο αρχείο επιβεβαίωσης καλεί την μέθοδο του smart contract checkData με μέθοδο call χωρίς καταναλώσει gas. Αυτή η μέθοδος του επιστρέφει τα αποτελέσματα που έχουν αποθηκευτεί στο smart contract. Στην συνέχεια, κάνει τους ελέγχους locally. Σε περίπτωση που δεν εντοπίσει λάθος, μπορεί να ολοκληρώσει την διαδικασία επιβεβαίωσης χωρίς να πληρώσει gas για να στείλει τα δικά του αποτελέσματα στο smart contract. Ωστόσο, σε περίπτωση εύρεσης λάθους, ξεκινάει η δεύτερη φάση. Σε αυτήν την φάση, ο client καλεί για κάθε task του αρχείου επιβεβαίωσης την μέθοδο updatePostedTaskPair με transact. Με αυτόν τον τρόπο αναγκάζει το smart contract να εκτελέσει τους ελέγχους και να ανανεώσει το status του app. Έπειτα, ξεκινάει το τρίτο στάδιο, στο οποίο ολόκληρος ο cluster μπορεί να δει το ενημερωμένο status της εφαρμογής. Επιπλέον, περιοδικά ο client μπορεί να τσεκάρει πόσα tasks έχει εκτελέσει ο κάθε worker και σε περίπτωση υπερβολικής χρήσης κάποιων worker να ενημερώσει τον cluster.

5.5 Ροή εκτέλεσης

Συνοψίζοντας, μια εφαρμογή ξεκινάει όταν ο χρήστης κάνει spark-submit την εργασία που θέλει στο spark. Στην συνέχεια, ο spark driver, ο οποίος τρέχει στο μηχάνημα του client, δημιουργεί το sparkContext. Έπειτα, ο sparkContext δημιουργεί το αρχείο επιβεβαίωσης και τους DAGScheduler και taskScheduler. Μετέπειτα , ο DAGScheduler δημιουργεί του ακυκλικούς γράφους και τα stages της εφαρμογής και εισάγει το κάθε task δύο φορές στα taskset. Ύστερα, ο DAGScheduler αναθέτει στον taskScheduler να δρομολογήσει τα taskset των stages. Ο taskScheduler επικοινωνεί με τον spark master για να βρει τους διαθέσιμους executors και, αφού ελέγξει ότι ίδια tasks δεν θα εκτελεστούν στον ίδιο worker, με την βοήθεια του taskset manager στέλνει τα tasks στους executors. Ο κάθε executor εκτελεί το task που του ανατέθηκε και μετά αφού δημιουργήσει την αντίστοιχη ηλεκτρον-

ική υπογραφή, στέλνει τα αποτελέσματα στον spark driver και στο smart contract. Ακολούθως, ο DAGScheduler συγκεντρώνει τα αποτελέσματα και αποθηκεύει τις ψηφιακές υπογραφές στο αρχείο επιβεβαίωσης. Τελικά, client μετά την ολοκλήρωση της εφαρμογής επικοινωνεί με το smart contract για την Διαδικασία Επιβεβαίωσης.



Σχήμα 5.1: Ροή εκτέλεσης

5.6 Εισαγωγή περισσότερων replicated tasks

Στην επέκταση του Apache Spark που αναλύθηκε παραπάνω, το κάθε task εισάγεται δύο φορές στο αντίστοιχο taskset. Ωστόσο, μπορεί γίνει η επιλογή μεγαλύτερου αριθμού αντιγραφής του κάθε task. Σε μια τέτοια περίπτωση οι παρεμβάσεις που αναλύθηκαν παραμένουν σε μεγάλο βαθμό ίδιες. Η διαφορά θα είναι ότι θα πρέπει να τροποποιηθεί το smart contract προκειμένου να κάνει τους ελέγχους για πλειάδες αντί για ζευγάρια. Επιπλέον, ανάλογα το replication number μπορεί να εφαρμοστούν δύο διαφορετικοί τρόποι ελέγχου:

1. strict έλεγχος: μια εφαρμογή χαρακτηρίζεται ως λανθασμένη μόλις βρεθεί ένα σφάλμα σε ένα αντίτυπο ενός task. Με αυτόν τον τρόπο απαιτείται να γίνει ξανά η εκτέλεση της εφαρμογής μόλις βρεθεί λάθος. Για διπλή αντιγραφή tasks ο strict έλεγχος είναι μονόδρομος.

2. non-strict έλεγχος: για μεγαλύτερους αριθμούς αντιγραφής, και κυρίως περιττούς, μπορεί να επιλεχθεί η απάντηση που εμφανίζεται περισσότερες φορές στην

πλειάδα αποτελεσμάτων. Με αυτόν τον τρόπο, ακόμα και αν υπάρχουν λάθος αποτελέσματα, μπορεί να συνεχιστεί η εκτέλεση της εφαρμογής με μεγαλύτερη πιθανότητα επιλογής της σωστής απάντησης. Προτιμάτε περιττός αριθμός αντιγραφής ώστε να υπάρχει πάντα πλειοψηφία.

Γενικά αναμένεται ότι όσο αυξάνεται ο αριθμός αντιγραφής των tasks τόσο αυξάνεται και η ανοχή σε σφάλματα και κατά συνέπεια η ανοχή σε κακόβουλους workers του συστήματος τόσο για τον strict όσο και για τον πιο χαλαρό έλεγχο. Αυτό συμβαίνει επειδή κάθε task εκτελείται περισσότερες φορές από διαφορετικούς workers και άρα είναι πιο σπάνιο να επιλεχθούν για όλα τα tasks κακόβουλοι workers. Ωστόσο, όσο περισσότερα tasks εισάγονται προς επεξεργασία, τόσο αναμένεται αύξηση στον χρόνο εκτέλεσης του κάθε app. Οι συνέπειες αύξησης του replication number θα μελετηθούν με πειραματική αξιολόγηση στο επόμενο κεφάλαιο.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

Στα πλαίσια της παρούσας Διπλωματικής εργασίας υλοποιήθηκε με βάση το μοντέλο που περιγράφηκε στο προηγούμενο κεφάλαιο μια demo επέκταση του spark μαζί με ένα smart contract. Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα των πειραμάτων και προσομοιώσεων που εκτελέστηκαν στο demo προκειμένου να αξιολογηθεί το εν λόγω μοντέλο.

6.1 Πειράματα

6.1.1 Περιβάλλον Εκτέλεσης

Προκειμένου να αξιολογηθεί η καθυστέρηση στον χρόνο εκτέλεσης που προκαλείται από την αντιγραφή των tasks αλλά και την επικοινωνία με το Smart Contract, στήθηκε ένας cluster από επτά workers οι οποίοι τρέχουν από ένα executor ο καθένας. Με άλλα λόγια, κάθε στιγμή ο cluster μπορεί να τρέξει μέχρι επτά ταυτόχρονα tasks. Επιπλέον, θα χρησιμοποιηθεί ένα απλό hadoop setup από το οποίο θα διαβάζουν τα αρχεία των πειραμάτων οι workers. Για το smart contract χρησιμοποιήθηκε το Ganache(`https://trufflesuite.com/ganache/`) το οποίο είναι μια εφαρμογή που επιτρέπει την δημιουργία ενός Ethereum blockchain τοπικά και διευκολύνει την διαδικασία ανάπτυξης και ελέγχου. Επιπρόσθετα, τα πειράματα θα τρέξουν σε τέσσερις διαφορετικές εκδοχές του spark. Η πρώτη θα είναι το απλό spark χωρίς καμιά επέκταση και θα χρησιμοποιηθεί σαν σημείο αναφοράς. Η δεύτερη θα είναι ένα spark που εισάγει για εκτέλεση δύο φορές το κάθε task ωστόσο δεν επικοινωνεί με το smart contract και ο έλεγχος γίνεται στον spark driver. Η τρίτη θα είναι πάλι ένα spark με διπλή εισαγωγή tasks που όμως επικοινωνεί με το smart contract για τον έλεγχο.

Η τελευταία έκδοση θα είναι ένα spark που εισάγει τρεις φορές το κάθε task και επικοινωνεί με το smart contract. Παρακάτω θα αναλυθούν τα πειράματα που θα εκτελεστούν.

6.1.2 Περιγραφή Πειραμάτων

Συνολικά θα εκτελεστούν τρία πειράματα.

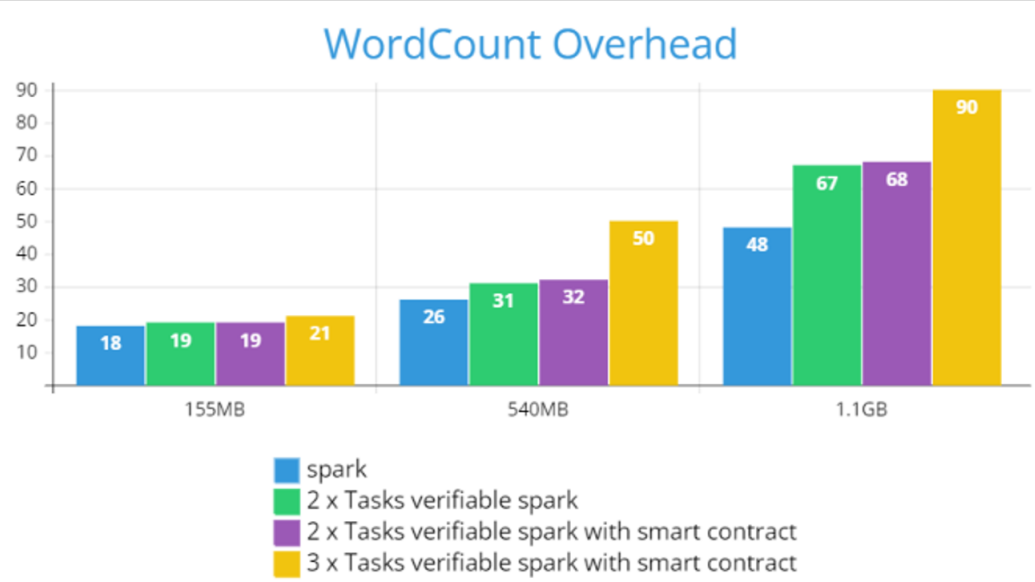
Το πρώτο είναι ένα απλό WordCount που μετράει για κάθε λέξη πόσες φορές εμφανίζεται στο κείμενο. Αυτό το πείραμα θα τρέξει τρεις φορές με είσοδο τρία αρχεία, ένα μικρό (155 MB), ένα μεσαίο(540 MB) και ένα μεγάλο(1.1 GB). Επιπλέον, κατά την διάρκεια εκτέλεσης, θα δημιουργηθούν για το μικρό αρχείο δύο stages με δύο tasks, για το μεσαίο δύο stages με πέντε tasks και για το μεγάλο δύο stages με εννιά tasks.

Για το δεύτερο πείραμα θα χρησιμοποιηθεί ένα dataset από το MovieLens(`https://grouplens.org/datasets/movielens/25m/`). Συγκεκριμένα, από αυτό το dataset θα χρησιμοποιηθούν δύο αρχεία, το movies.csv που έχει μέγεθος 2.89 MB και το ratings.csv με μέγεθος 646 MB. Το movies.csv περιέχει εγγραφές της μορφής movieId, title, genres, ενώ το ratings,csv περιέχει εγγραφές της μορφής userId, movieId, rating, timestamp. Τελικά στο δεύτερο πείραμα θα υπολογιστεί η μέση βαθμολογία για κάθε ταινία. Για να γίνει αυτό θα πραγματοποιηθεί ένα join στο movieId και ένα map και reduce. Ουσιαστικά, θα δημιουργηθούν δύο stages με επτά tasks το καθένα.

Στο τρίτο πείραμα θα χρησιμοποιηθούν ξανά τα αρχεία movies.csv και ratings.csv. Ωστόσο, σε αυτήν την περίπτωση θα υπολογιστεί για κάθε genre η ταινία με την μεγαλύτερη βαθμολογία και η ταινία με την μικρότερη βαθμολογία. Επιπλέον, σε περίπτωση ισοβαθμίας θα επιλέγεται η ταινία με τις περισσότερες βαθμολογίες. Είναι σαφές ότι αυτό το πείραμα είναι αρκετά πιο απαιτητικό σε υπολογιστικούς πόρους συγκριτικά με το προηγούμενο. Συγκεκριμένα, θα πρέπει αρχικά χωριστούν τα genres που μέχρι τώρα είναι ομαδοποιημένα για κάθε ταινία. Έπειτα θα πρέπει να υπολογιστεί για κάθε ταινία το πλήθος των βαθμολογιών της. Τελικά, πρέπει να πραγματοποιηθούν δύο joins και τρία MapReduce προκειμένου να παραχθεί το τελικό αποτέλεσμα. Συγκεκριμένα, σε αυτό το πείραμα δημιουργούνται δέκα stages εκ των οποίων το πρώτο έχει πέντε tasks, το δεύτερο έχει επτά tasks και όλα τα υπόλοιπα έχουν δώδεκα tasks. Ωστόσο, πρέπει να τονιστεί ότι δεν έχουν όλα τα stages το ίδιο υπολογιστικό βάρος. Με άλλα λόγια, τα τρία τελευταία stages έχουν σαν είσοδο δεδομένα από ReduceByKey με μικρό όγκο, με αποτέλεσμα να εκτελούνται πολύ γρηγορότερα από ότι τα προηγούμενα stages.

6.1.3 Παρουσίαση Αποτελεσμάτων

Το αποτέλεσμα του πρώτου πειράματος φαίνεται στο Σχήμα 6.1. Γενικά, βλέποντας το αποτέλεσμα γίνεται σαφές ότι όσο μεγαλύτερο το αρχείο τόσο μεγαλύτερη και η καθυστέρηση που προκαλείται από την αύξηση των αντιγραμμένων tasks. Με άλλα λόγια, ο χρόνος εκτέλεσης του spark χωρίζεται στον χρόνο έναρξης της εφαρμογής, στην δημιουργίας του πλάνου εκτέλεσης και στον χρόνο εκτέλεσης των tasks. Μέσα στον χρόνο εκτέλεσης των tasks περιέχεται και ο χρόνος μεταφοράς των δεδομένων. Επομένως, για μικρές εφαρμογές που απαιτούν λιγότερες μεταφορές και υπολογισμούς ανά task, ένα σημαντικό μέρος εκτέλεσης προέρχεται από την διαδικασία έναρξης και την δημιουργίας πλάνου εκτέλεσης. Ωστόσο, για πιο απαιτητικές εφαρμογές ο χρόνος καθορίζεται από την εκτέλεση των tasks και οι υπόλοιπες καθυστερήσεις τείνουν να γίνουν αμελητέες. Πιο συγκεκριμένα, για το μικρό αρχείο οι χρόνοι του απλού spark, του spark με διπλή αντιγραφή και τριπλή αντιγραφή είναι σχεδόν ίδιοι. Αυτό είναι αναμενόμενο δεδομένου ότι τα task του κάθε stage είναι δύο, στην περίπτωση διπλής αντιγραφής είναι τέσσερα και περίπτωση τριπλής αντιγραφής είναι έξι, ενώ τα tasks που μπορούν να τρέξουν ταυτόχρονα είναι επτά, άρα και στις τρείς περιπτώσεις όλα τα tasks τρέχουν παράλληλα. Ωστόσο, όσο μεγαλώνει το αρχείο εισόδου και κατα συνέπεια τα tasks ανα stage, τόσο δυσκολότερο είναι να εκτελεστούν παράλληλα τα tasks για μεγαλύτερο αριθμό αντιγραμμένων tasks. Αυτό συνεπάγεται σε μεγαλύτερη καθυστέρηση μεταξύ των εκδοχών του spark.



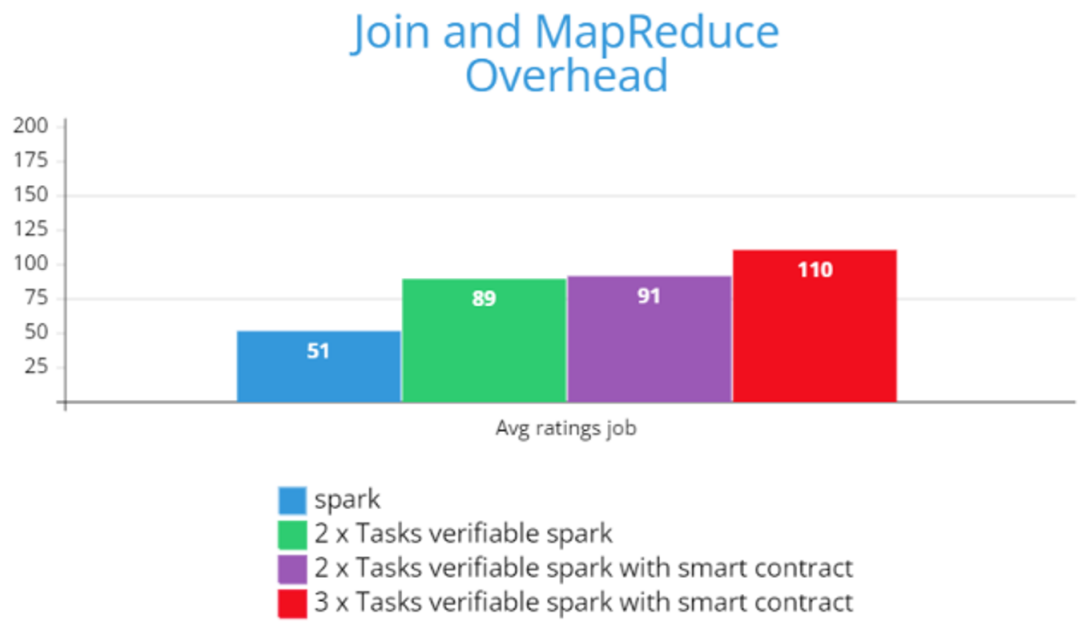
Σχήμα 6.1: Αποτέλεσμα πρώτου πειράματος

Τα αποτελέσματα των πειραμάτων δύο και τρία φαίνονται στα σχήματα 6.2 και 6.3 αντίστοιχα. Παρατηρώντας τα γραφήματα μπορούν να εξαχθούν αντίστοιχα συμπεράσ-

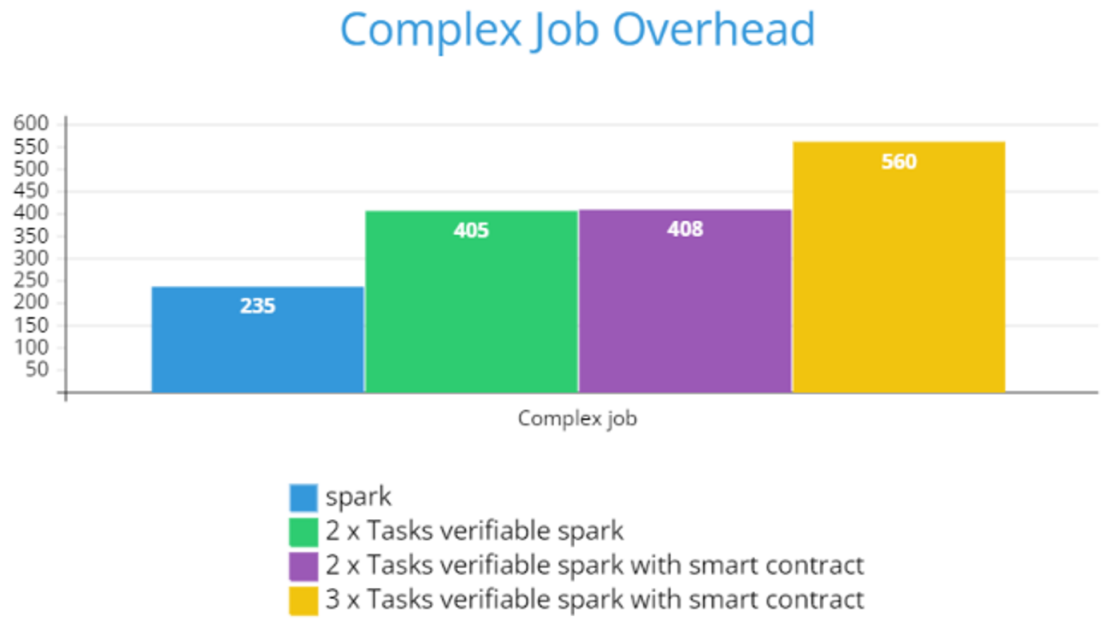
ματα με παραπάνω. Συγκεκριμένα, για το δεύτερο πείραμα οι χρόνοι εκτέλεσης παρουσιάζουν ομοιότητα με τους χρόνους του wordcount για το μεγάλο αρχείο. Παρόλα αυτά, το δεύτερο πείραμα έχει μεγαλύτερους χρόνους εκτέλεσης. Με την πρώτη ματιά αυτό μπορεί να φανεί λάθος δεδομένου ότι τα αρχεία εισόδου για το πείραμα δύο έχουν μέγεθος 2.89 MB και 646 MB σε αντίθεση με το 1.1 GB του πρώτου πειράματος. Επιπλέον, στο πρώτο πείραμα δημιουργούνται περισσότερα tasks από ότι στο δεύτερο. Ωστόσο, μπορεί να δικαιολογηθεί επειδή στο δεύτερο πείραμα πραγματοποιείται join που αποτελεί πιο χρονοβόρα πράξη από ότι ένα απλό MapReduce. Επομένως, το πείραμα δύο δείχνει ξανά ότι όσο πιο απαιτητικά είναι τα tasks τόσο καθορίζουν τον χρόνο εκτέλεσης και κατά συνέπεια και την επιπλέον καθυστέρηση για μεγαλύτερη αντιγραφή tasks.

Τελικά, στο τρίτο πείραμα, που αποτελεί και το πιο απαιτητικό υπολογιστικά, ο χρόνος καθορίζεται σε ακόμα μεγαλύτερο βαθμό από τον χρόνο εκτέλεσης των tasks. Για αυτό το λόγο, οι χρόνοι τείνουν να γίνουν ανάλογοι του αριθμού αντιγραφής των tasks. Με άλλα λόγια, ο χρόνο εκτέλεσης του spark με τριπλή εισαγωγή task έχει σχεδόν διπλάσιο overhead συγκριτικά με το spark με διπλή εισαγωγή task.

Τέλος, από όλα τα αποτέλεσμα φαίνεται ότι η επικοινωνία με τον smart contract δεν δημιουργεί επιπλέον καθυστέρηση.



Σχήμα 6.2: Αποτέλεσμα δεύτερου πειράματος



Σχήμα 6.3: Αποτέλεσμα τρίτου πειράματος

6.2 Προσομοιώσεις

Παρόλο που τα παραπάνω πειράματα οδήγησαν σε πολύ χρήσιμα συμπεράσματα, για την ολοκληρωμένη αξιολόγηση της συγκεκριμένης επέκτασης του spark πρέπει να μελετηθεί η κλιμακωσιμότητα και η ανοχή στα σφάλματα. Ωστόσο, δεν είναι εφικτή η πρόσβαση σε κάποιον μεγαλύτερο cluster, στον οποίο θα υπάρχουν κακόβουλοι workers με ρεαλιστική συμπεριφορά. Επομένως, για να ολοκληρωθεί η μελέτη θα γίνει προσέγγιση ενός τέτοιου cluster με διαδικασία προσομοίωση.

6.2.1 Περιγραφή Προσομοιώσεων

Συνολικά θα πραγματοποιηθούν τρεις προσομοιώσεις. Συγκεκριμένα, θα προσομοιωθεί ένας cluster από εκατό workers που τρέχουν έναν αριθμό από tasks.

Στη πρώτη προσομοίωση θα μελετηθεί πως επηρεάζει ο αριθμός εισαγωγής αντιγραμμένων tasks τον χρόνο εκτέλεσης μιας εφαρμογής. Συγκεκριμένα, θα προσομοιωθεί η δρομολόγηση χιλίων επί τον αριθμό αντιγραφής tasks σε 100 workers. Επισημαίνεται ότι αυτά τα tasks δρομολογούνται σαν να ανήκουν στο ίδιο stage.

Στην δεύτερη προσομοίωση, θα μελετηθεί πως επηρεάζεται η επιτυχημένη εύρεση λάθους από τον αριθμό αντιγραφής των tasks όταν χρησιμοποιείται strict έλεγχος.

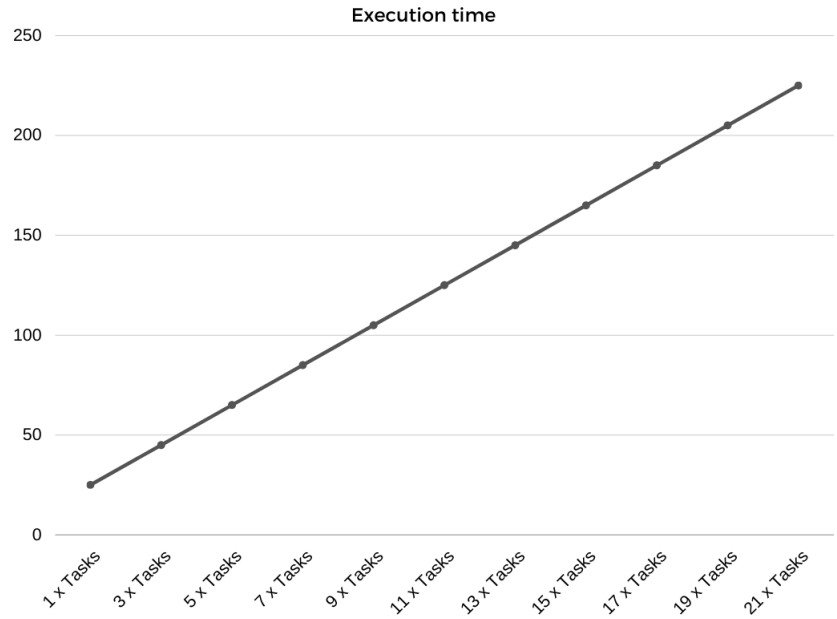
Συγκεκριμένα, θα προσομοιωθούν τρεις clusters με πέντε, δέκα και είκοσι κακόβουλους workers από τους συνολικούς εκατό. Στην συνέχεια, θα θεωρηθούν δύο εφαρμογές, μία με εκατό tasks και μία με χίλια tasks. Έπειτα, θα οριστεί ένας εύρος από αριθμούς αντιγραμμένων tasks που αρχίζει από το 3 και καταλήγει στο 21. Τελικά, για κάθε διαφορετική τριάδα cluster, εφαρμογής και αριθμού αντιγραφής πραγματοποιούνται χίλιες προσομοιώσεις και υπολογίζεται πόσες εφαρμογές εντοπίστηκε κάποιο λάθος. Σε αυτήν την διαδικασία προσομοίωσης γίνεται η υπόθεση ότι όλοι οι κακόβουλοι workers συνεργάζονται και κάνουν πάντα κοινά λάθοι όταν τους ανατίθενται ένα task. Είναι σαφές, ότι ένα τέτοιο σενάριο είναι πολύ δυσμενές και σε έναν πραγματικό cluster θα είναι πιο ευνοϊκές οι συνθήκες. Επιπλέον, επισημαίνεται ότι προκειμένου να μην εντοπιστεί λάθος πρέπει να δρομολογηθούν όλα τα αντίτυπα ενός task σε κακόβουλους workers.

Η τρίτη προσομοίωση μοιάζει πολύ με την δεύτερη. Ωστόσο, η μόνη αλλαγή της είναι ότι αντί να υπολογίζεται σε πόσες εφαρμογές εντοπίστηκε λάθος, υπολογίζεται το ποσοστό επιτυχούς επιλογής σωστής απάντησης με τον non strict έλεγχο. Με άλλα λόγια, υπολογίζεται σε πόσες εφαρμογές μπορεί με την πλειοψηφία των ίδιων αποτελεσμάτων να επιτευχθεί σωστό consensus και να συνεχίσει σωστά η εκτέλεση της εφαρμογής. Ουσιαστικά, προκειμένου μια εφαρμογή να επιτύχει σωστό consensus πρέπει πάνω από τα μισά αντίτυπα του κάθε task να αναθέτονται σε τίμιους workers.

6.2.2 Παρουσίαση Αποτελεσμάτων

Τα αποτελέσματα της πρώτης προσομοίωσης παρουσιάζονται στο Σχήμα 6.4. Όπως ήταν αναμενόμενο και από τα συμπεράσματα των πειραμάτων, φαίνεται ότι για εφαρμογές που έχουν πολλά tasks ανά stage και μεγάλο φόρτο εργασίας, ο χρόνος εκτέλεσης είναι ανάλογος του αριθμού αντιγραφής των tasks. Με άλλα λόγια, αυξάνοντας γραμμικά τον αριθμό replication των tasks προκαλεί γραμμική αύξηση και στον χρόνο εκτέλεσης. Επισημαίνεται ότι τα αποτελέσματα είναι εξωραϊσμένα λόγο της προσομοίωσης. Συγκεκριμένα, σε έναν πραγματικό cluster που τρέχει Apache Spark ο χρόνος εκτέλεσης προς τον αριθμό αντιγραφής των tasks δεν θα ήταν τέλεια ευθεία, ωστόσο θα έτεινε σε ευθεία.

Τα αποτελέσματα της δεύτερης προσομοίωσης παρουσιάζονται στο Σχήμα 6.5. Παρατηρείται ότι η αύξηση της αντιγραφής των tasks οδηγεί και σε μεγαλύτερα ποσοστά επιτυχούς εύρεσης λάθους. Συγκεκριμένα, ακόμα και για είκοσι κακόβουλους workers 9 αντιγραμμένα tasks αρκούν για να μηδενίσουν την πιθανότητα να περάσει κάποιο λάθος. Επιπλέον, μια ενδιαφέρουσα και λογική παρατήρηση είναι ότι όσο περισσότερα tasks έχει μια εφαρμογή τόσο πιθανότερο είναι να περάσει κάποιο λάθος, ειδικά για μικρότερους αριθμούς αντιγραφής. Αυτό συμβαίνει επειδή όσα περισσότερα tasks πρέπει να δρομολογηθούν τόσο αυξάνεται η πιθανότητα να πέσουν όλα τα αντίτυπα ενός task σε κακόβουλους workers.

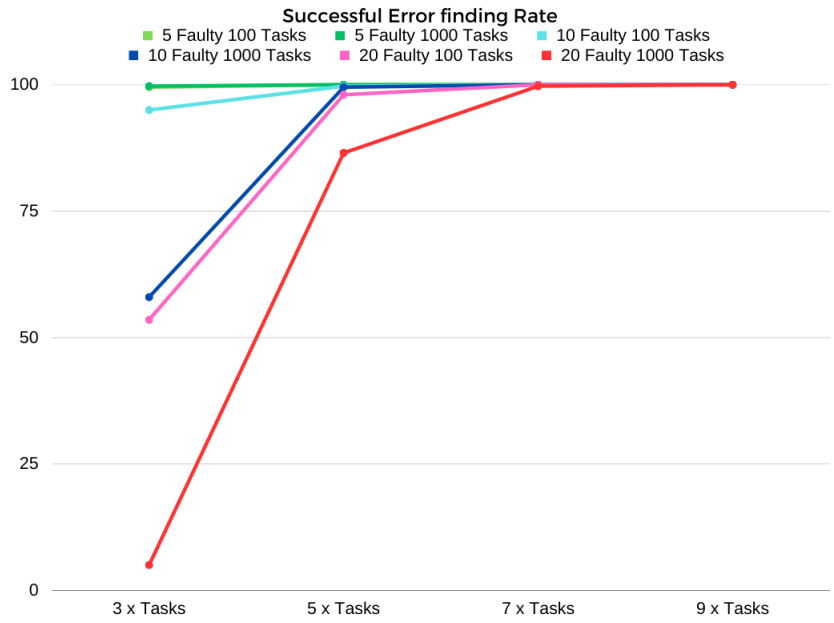


Σχήμα 6.4: Αποτέλεσμα πρώτης προσομοίωσης

Τα αποτελέσματα της τρίτης προσομοίωσης παρουσιάζονται στο Σχήμα 6.6. Τα συμπεράσματα της τρίτης προσομοίωση μοιάζουν αρκετά με τα συμπεράσματα της δεύτερης. Καταρχάς, και εδώ η αύξηση της αντιγραφής των tasks οδηγεί και σε μεγαλύτερα ποσοστά επιτυχούς consensus. Επιπλέον, τα περισσότερα tasks σε μια εφαρμογή οδηγούν σε μεγαλύτερη πιθανότητα λάθος consensus. Ωστόσο, η μεγάλη διαφορά συγκριτικά με το δεύτερο πείραμα είναι ότι στον non-strict έλεγχο απαιτείτε μεγαλύτερος αριθμός αντιγράφων προκειμένου τα ποσοστά consensus να γίνουν ικανοποιητικά. Συγκεκριμένα, παρατηρείται ότι για είκοσι faulty workers και την εφαρμογή με τα χίλια tasks το ποσοστό επιτυχία ανεβαίνει πολύ αργά και αρχίζει να ξεπερνά το 75% για 19 αντιγραμμένα tasks.

6.3 Συμπεράσματα

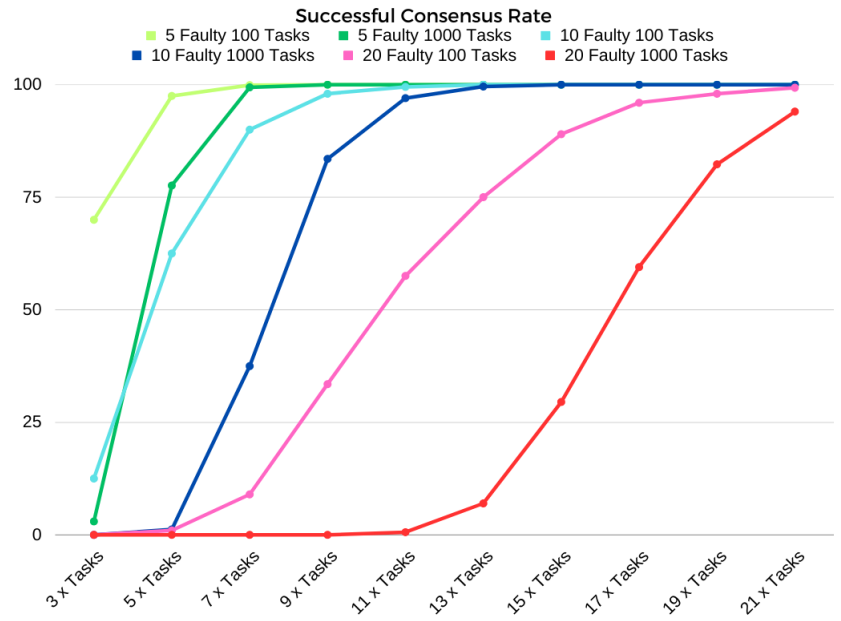
Από τα αποτελέσματα τόσο των πειραμάτων όσο και των προσομοιώσεων γίνεται σαφές ότι μεγαλύτερος αριθμός αντιγραμμένων tasks οδηγούν σε μεγαλύτερα ποσοστά ορθής επιβεβαίωσης της επεξεργασίας τόσο για τον strict όσο και για τον πιο χαλαρό



Σχήμα 6.5: Αποτέλεσμα δεύτερης προσομοίωσης

έλεγχο. Παρόλα αυτά, η αύξηση της αντιγραφής των tasks έχει αρκετά σημαντικό κόστος στον χρόνο εκτέλεσης της εφαρμογής όταν τα tasks ανά stage είναι περισσότερα των συνολικών executors. Αντιθέτως, για λίγα tasks ανά stage και μεγάλους clusters ο αριθμός της αντιγραφής των tasks μπορεί να είναι μεγαλύτερος χωρίς να επηρεάσει τόσο τον χρόνο εκτέλεσης. Επιπλέον, η επιλογή non strict ελέγχου για την επιβεβαίωση απαιτεί μεγαλύτερο αριθμό αντιγραφής. Ακόμα, όσο μεγαλώνει ο αριθμός των tasks μιας εφαρμογής τόσο μεγαλώνει και η πιθανότητα λάθους.

Επομένως, είναι δύσκολο να βρεθεί μία γενική λύση για όλους τους clusters. Ουσιαστικά, για κάθε cluster πρέπει να γίνει ανάλυση των απαιτήσεων προκειμένου να επιλεχθεί ένας αριθμός αντιγραφής που ταιριάζει στις ανάγκες του. Υπάρχουν πολλά κριτήρια που πρέπει να ληφθούν υπόψη για την επιλογή του κατάλληλου replication number. Καταρχάς, σημαντικό κριτήριο είναι ο αριθμός των κακόβουλων workers. Άρα, εξίσου σημαντικό κριτήριο είναι και το μέγεθος του cluster μιας και σε μεγαλύτερους clusters είναι αναμένονται περισσότεροι κακόβουλοι workers. Επιπλέον, σε μεγαλύτερους clusters η αύξηση του replication θα έχει μικρότερη επίδραση στον χρόνο εκτέλεσης συγκριτικά με μικρότερους cluster. Επιπρόσθετα, σημαντικό κριτήριο είναι το είδος των εφαρμογών και το πλήθος των tasks ανά stage που θα τρέχουν στο Spark. Επίσης, οι προσδοκίες των χρηστών για τα ποσοστά επιτυχίας



Σχήμα 6.6: Αποτέλεσμα τρίτης προσομοίωσης

μπορούν να επηρεάσουν τον αριθμό αντιγραφής. Τελικά, η επιλογή μεταξύ strict και non strict ελέγχου πρέπει να ληφθεί υπόψη για την επιλογή του αριθμού αντιγραφής. Συμπερασματικά, οι ανάγκες του κάθε cluster διαφέρουν, ωστόσο, γενικά είναι συνετή η επιλογή μεγαλύτερων αριθμών replication όσο μεγαλώνει το μέγεθος του κάθε cluster.

Κεφάλαιο 7

Συμπέρασμα και Μελλοντικές Επεκτάσεις

7.1 Συμπέρασμα

Στην παρούσα Διπλωματική εργασία επεκτάθηκε το Apache Spark προκειμένου να πραγματοποιείται replication των tasks και η αποστολή των αποτελεσμάτων τους σε ένα Smart Contract προκειμένου να γίνει επιβεβαίωση. Με αυτόν τον τρόπο επετεύχθη η ανάπτυξη μιας υλοποίησης του Apache Spark που μπορεί να τρέξει έμπιστα σε μη έμπιστους clusters. Επιπλέον, μέσω της πειραματικής αξιολόγησης παρατηρήθηκε ότι όσο μεγαλύτερος αριθμός replication τόσο μεγαλώνει και η ανοχή σε σφάλματα. Ωστόσο, με την αύξηση του αριθμού replication δημιουργείται και μεγαλύτερη καθυστέρηση στον χρόνο εκτέλεσης της εφαρμογής. Συγκεκριμένα, όσο περισσότερα είναι τα tasks ανα stage της εφαρμογής αναλογικά με το μέγεθος τους cluster τόσο μεγαλύτερη θα είναι και η καθυστέρηση. Επομένως, για λίγα tasks ανα stage η καθυστέρηση θα είναι αμελητέα ενώ για πολλά tasks ανα stage η καθυστέρηση θα είναι ανάλογη του αριθμού αντιγραφής. Τελικά, συγκεντρώνοντας τα παραπάνω, συμπεραίνεται ότι για μεγαλύτερους clusters που αναμένεται μεγάλος αριθμός μη έμπιστων κόμβων και μικρότερη καθυστέρηση από την αύξηση του αριθμού αντιγραφής πρέπει να επιλέγεται μεγαλύτερος αριθμός replication. Αντίθετα για μικρότερους clusters, πρέπει να γίνονται πιο συντηρητικές επιλογές μικρότερων αριθμών replication.

7.2 Μελλοντικές Επεκτάσεις

Η υλοποίηση που αναλύθηκε στα πλαίσια αυτής της διπλωματικής εργασίας αποτελεί ένα proof of concept. Επομένος, υπάρχουν αρκετά σημεία που χρήζουν βελτίωσης και αρκετές μελλοντικές επεκτάσεις που αν υλοποιηθούν μπορούν να εξελίξουν το σύστημα και να βελτιώσουν την επίδοση του. Παρακάτω θα αναφερθούν κάποιες επεκτάσεις που αξίζει να μελετηθούν.

7.2.1 Εκτέλεση πειραμάτων σε cluster με περισσότερους κόμβους

Για τα πειράματα που εκτελέστηκαν σε αυτήν την διπλωματική εργασία, χρησιμοποιήθηκε ένας cluster με επτά workers. Επιπλέον, για να επιτευχθεί και η μελέτη μεγαλύτερων clusters χρησιμοποιήθηκε η μέθοδος της προσομοίωσης. Ωστόσο, προκειμένου να συγκεντρωθούν ρεαλιστικά αποτελέσματα και να δημιουργηθεί μια έκδοση του συστήματος που να μπορεί να τρέξει αποδοτικά σε πραγματικές συνθήκες, πρέπει πρώτα να μελετηθεί η συμπεριφορά του σε πραγματικούς clusters με περισσότερους πόρους.

7.2.2 Βελτιστοποίηση του Smart Contract

Το Smart Contract που υλοποιήθηκε για αυτήν την εργασία αποτελεί ένα πρωτότυπο που πραγματοποιεί τους απαραίτητους ελέγχους και υπολογισμούς προκειμένου να είναι λειτουργική η επέκταση του Spark. Ωστόσο, για να χρησιμοποιηθεί για το verification ενός αληθινού cluster πρέπει να εκτελεί μερικούς ακόμα ελέγχους. Συγκεκριμένα, το smart contract πρέπει να ελέγχει από ποια διεύθυνση στάλθηκε η κάθε συναλλαγή και αν η διεύθυνση ανήκει σε κάποιον κόμβο του cluster. Οι παραπάνω έλεγχοι μπορούν να γίνουν με την χρήση των ψηφιακών υπογραφών που χρησιμοποιούνται ήδη στο Smart Contract. Επιπλέον, προκειμένου η επέκταση του spark που υλοποιήθηκε να είναι χρηστική και φτηνή, θα πρέπει να πραγματοποιηθεί ένα fine-tuning στο Smart Contract. Με άλλα λόγια, πρέπει να βελτιστοποιηθεί με σκοπό την ελαχιστοποίηση του κόστους σε gas των μεθόδων του. Επομένως, απαιτείται επιπλέον ανάπτυξη προκειμένου το smart contract να μπορεί να γίνει deploy στο Blockchain του Ethereum.

7.2.3 Master που συνεργάζεται με Workers

Στην περιγραφή της επέκτασης του Spark αναφέρθηκε ότι το Smart Contract αποθηκεύει στατιστικά σχετικά με το πόσα tasks έτρεξαν σε κάθε worker. Αυτό συμβαίνει προκειμένου να μπορεί να ελεγχθεί αν ο Spark Master προτείνει συχνότερα workers με τους οποίους συνεργάζεται. Αυτή η λειτουργικότητα του Smart Contract αλλά και η συνθήκη που πρέπει να ικανοποιηθεί προκειμένου να αποφασιστεί αν ο Master συνεργάζεται με συγκεκριμένους workers πρέπει να μελετηθεί περεταίρω και χρήζει βελτίωσης.

Βιβλιογραφία

[1] Jacek Laskowski. *The Internals of Apache Spark*. URL: `https://books.japila.pl/apache-spark-internals/`.

[2] Βουλγαρίδη Ιωάννη. *Επιβεβαιώσιμη Κατανεμημένη Επεξεργασία στο Apache Spark*. 2022. URL: `http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/18260`.

[3] Dr. Gavin Wood and Andreas M. Antonopoulos. *Mastering Ethereum*. 2018.