ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη επίδρασης της χρήσης κατανεμημένων συστημάτων γενικού σκοπού στην εκπαίδευση νευρωνικών δικτύων σε υπολογιστικό νέφος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Στέφανου Φ. Τζαβάρα

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη επίδρασης της χρήσης κατανεμημένων συστημάτων γενικού σκοπού στην εκπαίδευση νευρωνικών δικτύων σε υπολογιστικό νέφος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Στέφανου Φ. Τζαβάρα

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Μαρτίου 2021.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Ιωάννης Κωνσταντίνου
Καθηγητής Ε.Μ.Π.	Επικ. Καθηγητής Ε.Μ.Π.	Επικ. Καθηγητής
		Παν. Θεσσαλίας

Αθήνα, Μάρτιος 2021

.....

Στέφανος Φ. Τζαβάρας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο τομέας της βαθιάς μηχανικής μάθησης επεκτείνεται συνεχώς και ολοένα και αυξάνει τις εφαρμογές του, συμπεριλαμβανομένης και της κατηγοριοποίησης εικόνων. Για τον σκοπό αυτό αναπτύσσονται διαρκώς όλο και πιο πολύπλοκα νευρωνικά δίκτυα και αξιοποιούνται διάφορες τεχνικές προκειμένου να επιτευχθεί η επιθυμητή ακρίβεια στο μοντέλο. Όσο, όμως, αυξάνεται η πολυπλοκότητα των νευρωνικών δικτύων αλλά και ο όγκος των δεδομένων εκπαίδευσης, μεγαλώνει και η ανάγκη προς την κατανεμημένη εκπαίδευση νευρωνικών δικτύων προκειμένου να διαμοιραστεί ο φόρτος υπολογισμών και να επιτευχθεί υψηλότερη απόδοση και ταχύτητα. Στην παρούσα διπλωματική εργασία μελετώνται δύο δομές που υλοποιούν κατανεμημένη εκπαίδευση σε σύμπλεγμα (cluster) μηχανημάτων και αναλύονται εις βάθος προκειμένου να βρεθούν πλεονεκτήματα και μειονεκτήματα. Δίνεται βάση στο κατά πόσο ωφελεί ο συνδυασμός ενός κατανεμημένου συστήματος γενικού σκοπού με ένα ειδικό σύστημα βαθιάς μηχανικής μάθησης. Συνολικά γίνονται δύο πειράματα πάνω σε δύο διαφορετικά σύνολα εκπαίδευσης και νευρωνικά δίκτυα. Τα αποτελέσματα έδειξαν ότι συνολικά το κατανεμημένο TensorFlow είναι από μόνο του αποδοτικότερο και ταχύτερο απ' ότι το Spark σε συνδυασμό με την βιβλιοθήκη Keras του TensorFlow, χάρη στην ικανότητα ταχύτερης διαχείρισης και προετοιμασίας των επιμέρους μικρό-ομάδων σε κάθε βήμα εκπαίδευσης, κομμάτι στο οποίο το TensorFlow αποδείχθηκε ακόμα και 164X ταχύτερο. Παρ' όλα αυτά φάνηκε ότι η χρήση του Horovod για αυτήν την σύνδεση, προσέφερε σημαντική βελτίωση στην συλλογή και υπολογισμό των επιμέρους δεδομένων του κάθε μηχανήματος πάνω στην διαδικασία της εκπαίδευσης και αποδείχθηκε έως και 23X ταχύτερο.

Λέξεις κλειδιά: βαθιά μηχανική μάθηση, κατανεμημένη εκπαίδευση νευρωνικών δικτύων, επικοινωνία, σύμπλεγμα μηχανημάτων

Abstract

The field of deep learning is constantly expanding and increases more and more its applications, including that of image classification. For this purpose, even more complex neural networks are constantly being developed and various techniques are being exploited in order to achieve the desired model accuracy. However, the more the complexity and also the data volume of these neural networks are increased, the higher the need for distributed training exists in order to distribute the computation burden and achieve higher efficiency and speed. In this diploma thesis, two structures that implement distributed training in clusters of machines are being studied and thoroughly analyzed in order to define their advantages and disadvantages. Special attention is being given to the matter of whether it benefits to combine a distributed system of general purpose with a specialized one for deep learning. Two experiments in total are being held using two different datasets and neural networks. The results showed that, in general, the distributed implementation of TensorFlow is on its own more efficient and faster than the combination of Spark with the Keras library of TensorFlow, due to its capability of faster preparation of each training mini-batch, a part in which TensorFlow was proven even 164X faster. However, it came out that the use of Horovod for this connection and communication, offered significant improvement in the gathering and computing of each machine's gradients during the training process and was found even 23X faster.

Keywords: deep learning, distributed training of neural networks, communication, cluster

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή, κύριο Νεκτάριο Κοζύρη, καθώς επίσης και τον καθηγητή μου Ιωάννη Κωνσταντίνου, μέσω του οποίου εμπνεύστηκα το θέμα της διπλωματικής αυτής εργασίας.

Θα ήθελα, παράλληλα, να ευχαριστήσω θερμά τους φοιτητές και καθηγητές στο Εργαστήριο Υπολογιστικών Συστημάτων και συγκεκριμένα τον υποψήφιο διδάκτορα Νικόδημο Προβατά με τον οποίο συνεργαζόμουν καθ' όλη την διάρκεια σύνταξης της παρούσας εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω θερμά την οικογένειά μου, τους φίλους μου και τους συμφοιτητές μου για την διαρκή τους υποστήριξη σε όλα αυτά τα φοιτητικά μου χρόνια.

Πίνακας περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο της εργασίας

Η βαθιά μηχανική μάθηση αποτελεί πλέον σημαντικό κομμάτι σε εφαρμογές μεγάλων δεδομένων όπως στους τομείς της κατηγοριοποίησης εικόνων [1] ή της αναγνώρισης φωνής [2]. Συγκεκριμένα, συνεχώς αναπτύσσονται όλο και πιο μεγάλα και πολύπλοκα νευρωνικά δίκτυα προκειμένου να υποστηρίξουν τον μεγάλο όγκο δεδομένων. Ένα παράδειγμα αποτελεί η αρχιτεκτονική ResNet που κλήθηκε να χειριστεί το σύνολο δεδομένων ImageNet [3].

Φαίνεται, λοιπόν, ότι ο όγκος δεδομένων αλλά και η πολυπλοκότητα των αρχιτεκτονικών που χρησιμοποιούνται πλέον καθιστούν αναγκαία μια κατανεμημένη υλοποίηση εκπαίδευσης πάνω σε σύμπλεγμα μηχανημάτων. Ποικίλα συστήματα έχουν αναπτυχθεί προκειμένου να αντιμετωπίσουν αυτό το ζήτημα, όπως για παράδειγμα το TensorFlow της Google [4], το MXNet [5] και το PyTorch [6]. Διάφορες αρχιτεκτονικές και τεχνικές, επομένως, έχουν αναπτυχθεί για την υλοποίηση κατανεμημένης εκπαίδευσης. Μία πολύ συνήθης είναι αυτή του διακομιστή παραμέτρων (parameter server) [7]. Πάνω σε αυτήν την δομή, μπορούν να εφαρμοστούν τεχνικές όπως παραλληλισμός μοντέλου (model parallelism) [8], με την οποία παράμετροι των επιπέδων του δικτύου κατανέμονται σε πολλαπλούς διακομιστές παραμέτρων, η αντίστοιχα παραλληλισμός δεδομένων (data parallelism) [8] όπου τα δεδομένα εκπαίδευσης κατανέμονται στους επιμέρους «εργάτες»-μηχανήματα του cluster. Επιπλέον, χρησιμοποιείται ευρέως, πλέον, και η τεχνική του AllReduce [9] όπου κάθε κόμβος διατηρεί το αποτέλεσμα από το κομμάτι υπολογισμού που έχει εκτελέσει και έπειτα όλοι οι κόμβοι επικοινωνούν και συγκεντρώνουν τα αποτελέσματά τους προκειμένου να συγχρονιστούν και να προχωρήσουν στο επόμενο βήμα.

Αντίστοιχα, όμως, υπάρχουν πολλά διαθέσιμα εξειδικευμένα κατανεμημένα συστήματα που αποσκοπούν στην αποδοτική διαχείριση μεγάλων δεδομένων και ροών δεδομένων. Τέτοια παραδείγματα αποτελούν το Apache Hive [10], το Impala [11], το Presto [12] και το GraphLab [13]. Προς την ίδια κατεύθυνση κινήθηκε και το Apache Spark [14], ένα σύστημα γενικού σκοπού το οποίο συνδυάζει πολλές διαφορετικές υλοποιήσεις και λειτουργικότητες και επεκτείνεται και προς τον τομέα της μηχανικής μάθησης με βιβλιοθήκες όπως η MLlib [15], χωρίς όμως ακόμα να υποστηρίζει αρχιτεκτονικές βαθιάς μηχανικής μάθησης.

Παίρνοντας, επομένως, αυτά ως δεδομένα μπορούν να τεθούν ερωτήματα που αφορούν το πώς θα επιτευχθεί αποδοτικότερη κατανεμημένη εκπαίδευση που θα αντέχει και υποστηρίζει όλο και πιο πολύπλοκα δίκτυα και μεγάλα δεδομένα. Σκοπός, λοιπόν, αυτής της διπλωματικής εργασίας είναι να διερευνηθεί το κατά πόσο ο συνδυασμός ενός εξειδικευμένου συστήματος βαθιάς μάθησης, όπως το TensorFlow, και ενός γενικού σκοπού, όπως το Spark, μπορεί να προσφέρει βελτίωση και ενδιαφέρουσες επεκτάσεις στο ζήτημα της κατανεμημένης εκπαίδευσης νευρωνικών δικτύων.

1.2 Δομή της εργασίας

Η διάρθρωση της παρούσας διπλωματικής εργασίας έχει ως εξής:

- Στο **Κεφάλαιο 2** πραγματοποιείται μία βιβλιογραφική περιγραφή εννοιών που αφορούν τα νευρωνικά δίκτυα, την διαδικασία εκπαίδευσής τους καθώς και κατανημένες υλοποιήσεις.

- Στο **Κεφάλαιο 3** αναλύεται η αρχιτεκτονική και ο τρόπος λειτουργίας των συστημάτων που χρησιμοποιήθηκαν στο πειραματικό μέρος.

- Στο **Κεφάλαιο 4** περιγράφεται η διαδικασία και οι φάσεις των πειραμάτων που διεκπεραιώθηκαν στα πλαίσια της διπλωματικής εργασίας.

- Στο **Κεφάλαιο 5** παρουσιάζονται τα αποτελέσματα που προέκυψαν από τα πειράματα και γίνεται εκτενής σχολιασμός και ανάλυσή τους.

- Τέλος στο **Κεφάλαιο 6** γίνεται μία σύνοψη των συμπερασμάτων που προέκυψαν από τον σχολιασμό των αποτελεσμάτων καθώς προτείνονται πιθανές επεκτάσεις αυτής της εργασίας.

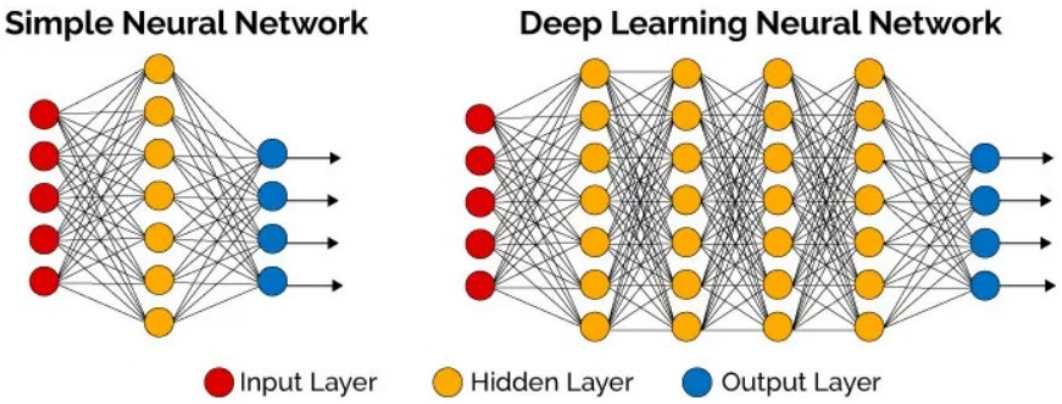
Κεφάλαιο 2

Νευρωνικά Δίκτυα

Στο κεφάλαιο αυτό θα πραγματοποιηθεί μία βιβλιογραφική περιγραφή των νευρωνικών δικτύων. Αρχικά θα παρατεθούν γενικές πληροφορίες και θα σχολιαστούν τόσο η μορφή όσο και η δομή των τροφοδοτικών νευρωνικών δικτύων. Έπειτα θα γίνει μία περιγραφή της διαδικασίας εκπαίδευσης των νευρωνικών δικτύων μαζί με τα επιμέρους τμήματα αυτής της λειτουργικότητας. Παρακάτω θα δοθεί έμφαση σε ένα συγκεκριμένο είδος, τα συνελικτικά νευρωνικά δίκτυα, τα οποία θα χρησιμοποιηθούν στο πειραματικό μέρος της παρούσας διπλωματικής εργασίας. Τέλος, θα αναλυθούν τεχνικές και αρχιτεκτονικές που αποσκοπούν στην κατανεμημένη εκπαίδευση νευρωνικών δικτύων. Σημειώνεται ότι αρκετές πληροφορίες αφορούν κυρίως τα προβλήματα κατηγοριοποίησης καθώς σε αυτά επικεντρώνεται στην διπλωματική εργασία.

2.1 Γενικές Πληροφορίες Περί Τροφοδοτικών Νευρωνικών Δικτύων

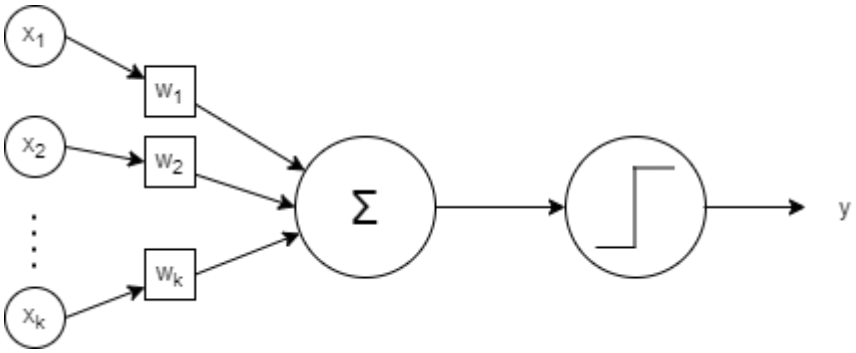
Σκοπός των μεθόδων βαθιάς μηχανικής μάθησης είναι η εκμάθηση χαρακτηριστικών σε ιεραρχίες [16]. Έτσι, τα χαρακτηριστικά στα υψηλότερα επίπεδα σχηματίζονται από την σύνθεση χαρακτηριστικών χαμηλότερων επιπέδων. Τέτοιες αρχιτεκτονικές, λοιπόν, εκμεταλλεύονται την ιεραρχική δομή και τις κρυφές μεταβλητές προς την επίλυση διαφόρων προβλημάτων, ιδιαιτέρως επιβλεπόμενης μάθησης [17]. Σε γενικές γραμμές, το μοντέλο αποτελείται από μια αλληλουχία νευρώνων οργανωμένων σε επίπεδα, όπου η πληροφορία μεταφέρεται ανάμεσα στα επίπεδα μέχρι να βγει στο τέλος το επιθυμητό αποτέλεσμα στην έξοδο. Λόγω της θεωρητικής έλξης τους αλλά και της αναλογίας αυτών των εννοιών με την βιολογία και την λειτουργία των εγκεφαλικών νευρώνων, τα πολυεπίπεδα τροφοδοτικά νευρωνικά δίκτυα είναι πολύ δημοφιλή μοντέλα.



Εικόνα 2.1 Παραδείγματα τροφοδοτικών νευρωνικών δικτύων

Στην Εικόνα 2.1 [18] φαίνονται δύο παραδείγματα νευρωνικών δικτύων αυτής της αρχιτεκτονικής. Η πληροφορία μεταφέρεται αλυσιδωτά από επίπεδο σε επίπεδο όπου η έξοδος του ενός γίνεται είσοδος του επόμενου. Τα επίπεδα του δικτύου διαχωρίζονται στο **επίπεδο εισόδου**, όπου δίνεται το αντικείμενο προς κατηγοριοποίηση αντιστοιχώντας νευρώνες με χαρακτηριστικά, τα **κρυφά επίπεδα**, που αποσκοπούν στην περαιτέρω επεξεργασία των δεδομένων εισόδου και το **επίπεδο εξόδου** όπου παράγεται το επιθυμητό αποτέλεσμα στην μορφή που αρμόζει στο εκάστοτε πρόβλημα.

Κάθε νευρώνας του δικτύου που αναλύθηκε παραπάνω, μπορεί να θεωρηθεί ως ένας απλός Αισθητήρας (Perceptron) [19]. Βάση του Αισθητήρα αποτελεί το μαθηματικό μοντέλο ενός νευρώνα των McCulloch και Pitts, το οποίο έχει την δομή που φαίνεται στην Εικόνα 2.2.



Εικόνα 2.2: Το δομικό στοιχείο Perceptron

Οι είσοδοι x_i πολλαπλασιάζονται με τα βάρη w_i, έπειτα ο νευρώνας αθροίζει αυτές τις τιμές όπως φαίνεται στην εξίσωση 2.1:

$$\boldsymbol{h = \sum_{i=1}^{k} w_i x_i} \tag{2.1}$$

και τέλος το αποτέλεσμα αυτό περνάει σαν είσοδος σε μια συνάρτηση που ονομάζεται **συνάρτηση ενεργοποίησης** και καθορίζει αν ο νευρώνας θα συνεισφέρει στο τελικό αποτέλεσμα ή όχι. Η πιο απλή συνάρτηση ενεργοποίησης είναι η βηματική συνάρτηση κατωφλίου όπου ένα κατώφλι θ καθορίζει την ενεργοποίηση του νευρώνα. Ενδεικτικά, οι πιο ευρέως χρησιμοποιούμενες συναρτήσεις ενεργοποίησης (για το επίπεδο εισόδου και τα κρυφά επίπεδα) είναι η **σιγμοειδής**

(sigmoid) [20], η **υπερβολική εφαπτομένη (tanh)** [21] και η **μονάδα γραμμικού ανορθωτή (rectified lineart unit – ReLU)** [22]**.**

Όσον αφορά το επίπεδο εξόδου, η συνηθέστερη συνάρτηση ενεργοποίησης στα προβλήματα κατηγοριοποίησης είναι η **Softmax.**

2.2 Συνελικτικά Τροφοδοτικά Νευρωνικά Δίκτυα

Στην ενότητα αυτή πραγματοποιείται μία βιβλιογραφική ανασκόπηση της αρχιτεκτονικής των συνελικτικών νευρωνικών δικτύων.

Τα συνελικτικά τροφοδοτικά νευρωνικά δίκτυα [23] είναι ένας τύπος μοντέλων βαθιάς μηχανικής μάθησης που ειδικεύονται στην επεξεργασία δεδομένων που έχουν μορφή πλέγματος. Στις περισσότερες περιπτώσεις πρόκειται για οπτικό υλικό όπως εικόνες και είναι σχεδιασμένα να εκπαιδεύονται και να μαθαίνουν ιεραρχικά χωρικά χαρακτηριστικά. Τα κρυφά επίπεδα αυτών των δικτύων περιγράφουν αρκετά περίπλοκες συναρτήσεις και συνήθως για συνάρτηση ενεργοποίησης χρησιμοποιούν την ReLU. Τα κύρια δομικά στοιχεία της αρχιτεκτονικής αυτής είναι τα εξής:

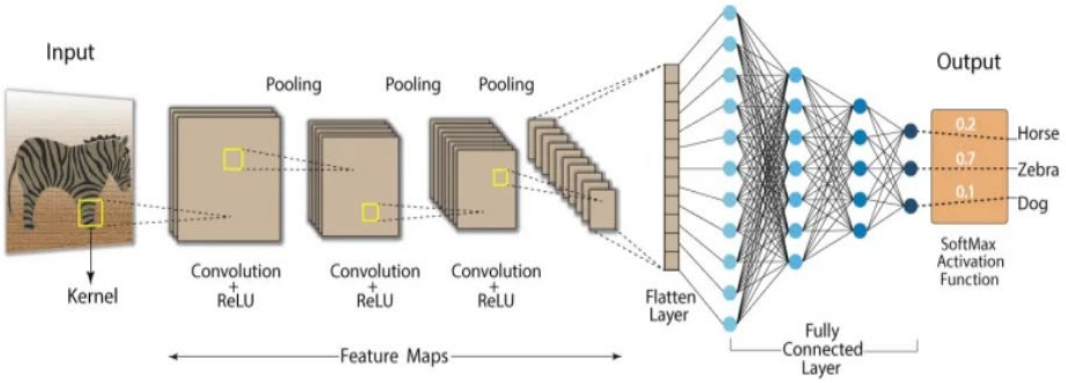
- **Συνελικτικό (Convolutional):** Βασικό επίπεδο αυτών των δικτύων που εκτελεί εξαγωγή χαρακτηριστικών με τη χρήση συνελικτικού τελεστή. Σε αυτό το επίπεδο η μόνη υπερπαράμετρος προς εκπαίδευση είναι το μέγεθος του συνελικτικού πυρήνα. Άλλες υπερπαράμετροι αφορούν τα κανάλια εισόδου και εξόδου. Οφέλη του συνελικτικού τελεστή περιλαμβάνουν, μεταξύ άλλων, την σύλληψη όλο και μεγαλύτερου πεδίου όψης όταν συνδυάζεται με υπο-δειγματοληψία (εξηγείται παρακάτω) και μεγαλύτερη αποδοτικότητα μοντέλου σε σχέση με τα πλήρως συνδεδεμένα νευρωνικά δίκτυα καθώς χρειάζεται μόνο μία υπερπαράμετρος προς εκπαίδευση.

- **Χωρικής Υπο-δειγματοληψίας (Pooling):** Το επίπεδο αυτό παρέχει υπο-δειγματοληψία με σκοπό να μειώσουν τις διαστάσεις των χαρακτηριστικών. Μια συνήθης μορφή τέτοιας διεργασίας είναι το max pooling όπου από μία ομάδα νευρώνων επικρατεί μόνο το μέγιστο. Αντίστοιχα, μπορεί να χρησιμοποιηθεί average pooling όπου εφαρμόζεται τελεστής μέσου όρου αντί για μέγιστο. Η χωρική υπο-δειγματοληψία μπορεί να εφαρμοστεί τοπικά σε μια ομάδα νευρώνων του επιπέδου ή καθολικά σε όλους τους νευρώνες του επιπέδου.

- **Πλήρως Συνδεδεμένο (Fully Connected):** Αντίστοιχο με τα επίπεδα ενός απλού τροφοδοτικού νευρωνικού δικτύου, επίσης γνωστό ως πυκνό επίπεδο (dense layer). Παράγει στην έξοδο το τελικό αποτέλεσμα που σε προβλήματα κατηγοριοποίησης είναι η πιθανότητα ένταξης του αντικειμένου στην κάθε κλάση και συνήθης είναι η Softmax για συνάρτηση ενεργοποίησης.

Στην εκπαίδευση νευρωνικών δικτύων υπάρχει ένας κίνδυνος που ονομάζεται υπερπροσαρμογή (overfitting)**.** Αυτό συμβαίνει όταν το μοντέλο μαθαίνει μερικά μοτίβα πολύ συγκεκριμένα για τα δεδομένα εκπαίδευσης με αποτέλεσμα να μην αποδίδει καλά σε επόμενα καινούρια δεδομένα και έτσι το μοντέλο δεν είναι γενικευμένο όπως πρέπει. Για να αποφευχθεί αυτό το φαινόμενο χρησιμοποιούνται συνήθως στα συνελικτικά νευρωνικά δίκτυα επίπεδα **απόσυρσης (dropout)** και **κανονικοποίησης (normalization).** Στο επίπεδο απόσυρσης, επιλέγονται τυχαία κάποιοι νευρώνες και νεκρώνονται (τίθεται ενεργοποίηση 0) έτσι ώστε το μοντέλο να είναι λιγότερο ευαίσθητο σε συγκεκριμένα βάρη μέσα στο δίκτυο. Το επίπεδο κανονικοποίησης είναι ένα συμπληρωματικό επίπεδο που κανονικοποιεί τις τιμές εισόδου στο επόμενο επίπεδο και μειώνει τον κίνδυνο υπέρ-προσαρμογής, προσδίδει μεγαλύτερα ποσοστά μάθησης και μειώνει την εξάρτηση από

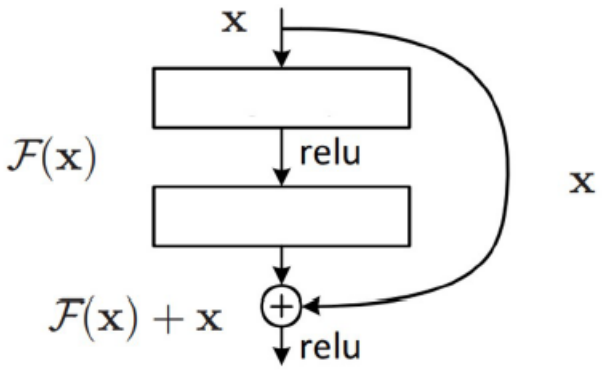
την αρχικοποίηση των παραμέτρων του μοντέλου. Στην Εικόνα 2.3 [24] φαίνεται ένα παράδειγμα συνελικτικού νευρωνικού δικτύου με τα επιμέρους επίπεδά του.



Εικόνα 2.3: Παράδειγμα συνελικτικού νευρωνικού δικτύου

Ένα σημαντικό είδος συνελικτικού νευρωνικού δικτύου που αξιοποιείται και στην συνέχεια στο πειραματικό μέρος είναι τα υπολειπόμενα νευρωνικά δίκτυα ResNet [3].Το είδος αυτό είναι μια αρχιτεκτονική που εκμεταλλεύεται την τεχνική των υπολοίπων για να προσεγγίσει περίπλοκες συναρτήσεις. Θεωρώντας x την είσοδο στο επίπεδο, H(x) μία περίπλοκη συνάρτηση που θεωρητικά προσομοιώνεται από αυτό το επίπεδο και F(x) μία συνάρτηση προσέγγισης, μπορεί να γίνει η παραδοχή της εξίσωσης 2.2 :

$$\boldsymbol{H(x) = F(x) + x} \tag{2.2}$$



Εικόνα 2.4: Το δομικό στοιχείο ενός δικτύου ResNet

Στην Εικόνα 2.4 απεικονίζεται η εξίσωση που αφορά ένα τμήμα νευρωνικού δικτύου. Τα ενδιάμεσα δομικά στοιχεία αποτελούνται κυρίως από συνελικτικά δίκτυα που ανάλογα την αρχιτεκτονική διαφέρουν σε πλήθος, μέγεθος χαρακτηριστικών και αλληλουχίας. Σημαντικό στοιχείο της αρχιτεκτονικής ResNet είναι ότι δεν προσθέτει επιπλέον παραμέτρους προς εκπαίδευση και άρα η χρήση τους δεν αυξάνεται την πολυπλοκότητα σχετικά με άλλες αρχιτεκτονικές.

2.3 Εκπαίδευση Νευρωνικών Δικτύων

Στην ενότητα αυτή θα γίνει αναλυτική περιγραφή της διαδικασίας της εκπαίδευσης νευρωνικών δικτύων. Πιο συγκεκριμένα, θα γίνει βιβλιογραφική αναφορά στην έννοια της βελτιστοποίησης και των συναρτήσεων βελτιστοποίησης που χρησιμοποιούνται συνήθως και έπειτα θα γίνει παράθεση των βημάτων της εκπαίδευσης.

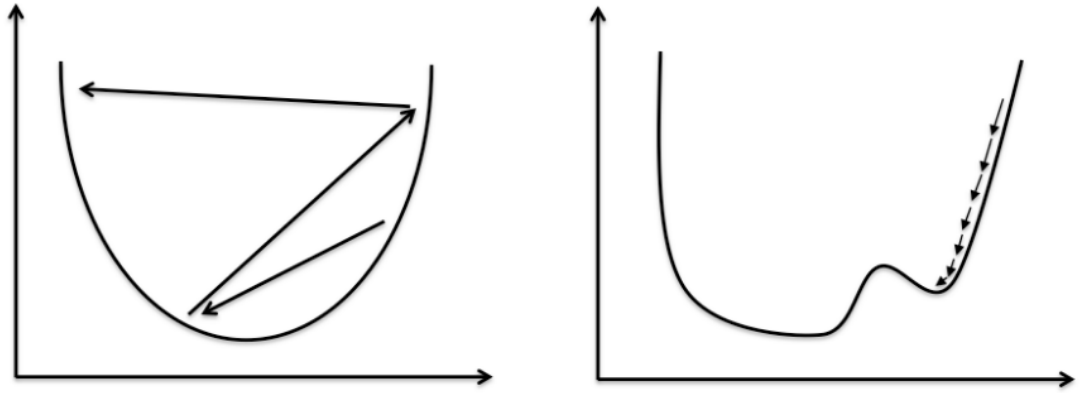
2.3.1 Βελτιστοποιητές

Σε γενικές γραμμές, η εκπαίδευση νευρωνικών δικτύων βασίζεται σε μία απλή διαδικασία όπου το δίκτυο βγάζει σαν αποτέλεσμα την πρόβλεψή του για το εκάστοτε πρόβλημα και έπειτα μέσω μιας συνάρτησης που ονομάζεται συνάρτηση κόστους, εκτιμώνται οι ενημερώσεις που πρέπει να γίνουν στις παραμέτρους του δικτύου προκειμένου να μειωθεί η απόκλιση από την πρόβλεψη και το πραγματικό αναμενόμενο αποτέλεσμα. Ειδικότερα, στα προβλήματα κατηγοριοποίησης η συνάρτηση κόστους που χρησιμοποιείται ευρέως ονομάζεται **κατηγορική εγκάρσια εντροπία (categorical cross-entropy)** [25]**.** Συνοπτικά, αυτό που επιτυγχάνει είναι να μετρήσει την απόκλιση του αποτελέσματος που προκύπτει από την συνάρτηση Softmax του επιπέδου εξόδου ως προς την κατηγορία που ανήκει πραγματικά το αντικείμενο εισόδου.

Έτσι, λοιπόν, προκύπτει το ζήτημα με ποιον αλγόριθμο θα πραγματοποιείται η βελτιστοποίηση της συνάρτησης κόστους. Από τους πιο θεμελιώδεις και δημοφιλείς αλγορίθμους είναι ο αλγόριθμος καθόδου κλίσεων (Gradient Descent – GD) [26]. Βασική ιδιότητα του αλγορίθμου αυτού είναι ότι είναι επαναληπτικός και σκοπός του είναι να συγκλίνει στο ελάχιστο (ενδεχομένως κάποιες φορές σε τοπικό αντί για ολικό) της συνάρτησης κόστους χρησιμοποιώντας την κλίση (gradient) της συνάρτησης κόστους ως προς τι παραμέτρους του δικτύου προκειμένου να επιτευχθεί η επιθυμητή σύγκλιση. Μία βασική παράμετρος του αλγορίθμου αυτού είναι ο ρυθμός μάθησης (learning rate - η) η οποία καθορίζει την ταχύτητα σύγκλισης του αλγορίθμου προς το ελάχιστο σε κάθε επανάληψη. Στην εξίσωση 2.3 φαίνεται η μαθηματική έκφραση του αλγορίθμου που εφαρμόζεται σε κάθε επανάληψη, όπου L είναι η συνάρτηση κόστους και με ~ ορίζεται η πρόβλεψη του δικτύου για την κλάση του αντικειμένου.

$$\vec{w}_{i+1} = \vec{w}_i - \frac{\eta}{k} \cdot \sum_{j=1}^{k} \nabla_{\vec{w}_i} L(y_j, \tilde{y}_j) \tag{2.3}$$

Καθοριστικό ρόλο στην εξέλιξη του αλγορίθμου παίζει ο ρυθμός μάθησης. Στην Εικόνα 2.5 [27] φαίνεται η επίδραση που έχει ανάλογα την τιμή του. Όταν ο ρυθμός μάθησης είναι αρκετά μεγάλος, γίνονται μεν μεγάλα βήματα προς την σύγκλιση στο ελάχιστο αλλά μπορεί σε μερικές περιπτώσεις να γίνεται απομάκρυνση στιγμιαία, αυξάνοντας εν τέλει τον χρόνο που απαιτείται για σύγκλιση. Από την άλλη μεριά, όταν είναι πολύ μικρός ο ρυθμός μάθησης, η κατεύθυνση είναι μεν πάντα σωστή προς το σημείο βελτιστοποίησης αλλά το βήματα είναι πολύ μικρά οπότε και πάλι καθυστερεί η σύγκλιση. Μάλιστα γίνεται φανερός ο κίνδυνος εγκλωβισμού σε κάποιο τοπικό αντί για ολικό ελάχιστο. Καθίσταται, συνεπώς, μεγάλη η ανάγκη εύρεσης της ιδανικής τιμής του.



(α) Μεγάλος ρυθμός μάθησης (β) Μικρός ρυθμός μάθησης

Εικόνα 2.5: Επίδραση του ρυθμού μάθησης

Ο αλγόριθμος Gradient Descent, βέβαια, μπορεί να τροποποιηθεί με διάφορες τρόπους και προσφέρει μερικές παραλλαγές όπως ο Στοχαστικός Αλγόριθμος Καθόδου Κλίσεων (Stochastic Gradient Descent – SGD) [28] και ο Στοχαστικός Αλγόριθμος Καθόδου Κλίσεων Μικρό-ομάδων (Mini-Batch Stochastic Gradient Descent, Mini-Batch SGD).

Όσον αφορά τον SGD, η βασική του διαφορά με τον απλό GD είναι ότι σε κάθε βήμα των επαναλήψεων χρησιμοποιείται μόνο ένα από τα δεδομένα εκπαίδευσης που επιλέγεται τυχαία και όχι όλο το σύνολο των δεδομένων, προσφέροντας μερικές βελτιώσεις σε σχέση με τον απλό αλγόριθμο καθόδου κλίσεων. Καταρχάς, ο SGD επιταχύνει αρκετά τους υπολογισμούς κάθε βήματος εκπαίδευσης καθώς δεν εφαρμόζεται πάνω στο σύνολο των δεδομένων αλλά πάνω σε μόνο ένα σημείο. Επιπλέον, ο απλός GD έχει το μειονέκτημα ότι μπορεί να «εγκλωβιστεί» σε κάποιο τοπικό ελάχιστο ενώ ο SGD, παρότι εισάγει μια διασπορά στις τιμές της συνάρτησης κόστους, μπορεί να ξεφύγει από αυτό και να συγκλίνει εν τέλει στο πραγματικό βέλτιστο σημείο χάρη στην τυχαιότητα επιλογής σημείου στο κάθε βήμα εκπαίδευσης.

Από την άλλη, ο Mini-Batch SGD συνδυάζει αυτές τις δύο υλοποιήσεις. Ουσιαστικά σε κάθε βήμα της εκπαίδευσης πραγματοποιεί ενημέρωση παραμέτρων παίρνοντας μια μικρό-ομάδα από τα δεδομένα εκπαίδευσης αντί για το σύνολο. Είναι ευρέως διαδεδομένη τεχνική και επιτυγχάνει αύξηση αποδοτικότητας μειώνοντας την διασπορά στις τιμές της συνάρτησης κόστους και αυξάνοντας την ταχύτητα σύγκλισης.

Παρά τις προσαρμογές και τις βελτιώσεις, όμως, στον αλγόριθμο GD, υπάρχει μία σημαντική πρόκληση στην εκπαίδευση και την ελαχιστοποίηση της συνάρτησης κόστους. Αυτή είναι η επιλογή ενός σωστού και αποδοτικού ρυθμού μάθησης που θα επιτυγχάνει ταχεία σύγκλιση, δεν θα προσδίδει μεγάλη διασπορά στις τιμές και δεν θα εγκλωβίζεται σε τοπικά ελάχιστα. Η επιλογή αυτή εκ των προτέρων είναι πολύ δύσκολη υπόθεση και συχνά αδύνατη.

Για αυτόν τον λόγο χρησιμοποιούνται ευρέως μερικές βελτιώσεις των αλγορίθμων τύπου GD που βασίζονται στην προσαρμογή του ρυθμού μάθησης σε κάθε βήμα της επαναληπτικής διαδικασίας. Συνοπτικά αναλύονται δύο σημαντικοί εξ αυτών.

Ο πρώτος ονομάζεται Αλγόριθμος Προσαρμοσμένης Κλίσης (Adaptive Gradient Algorithm - Adagrad) [29]. Ο αλγόριθμος αυτός προσαρμόζει τον ρυθμό μάθησης στις παραμέτρους πραγματοποιώντας μεγαλύτερες ενημερώσεις για λιγότερο συχνές και μικρότερες ενημερώσεις για περισσότερο συχνές παραμέτρους. Σε κάθε χρονικό βήμα t, επομένως, έχουμε διαφορετικό ρυθμό μάθησης που βασίζεται στα προγενέστερα διανύσματα κλίσης. Έχουμε λοιπόν στην εξίσωση 2.4:

$$\vec{w}_{i+1} = \vec{w}_i - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot \nabla_{\vec{w}_i} L(\vec{w}_i, \vec{x}_j, y_j) \quad (2.4)$$

όπου G_i ορίζεται ένας διαγώνιος πίνακας όπου κάθε στοιχείο της διαγωνίου (k,k) περιέχει το άθροισμα των τετραγώνων των κλίσεων της παραμέτρου με σειρά k μέχρι το χρονικό βήμα i και το ε είναι ένας όρος εξομάλυνσης της τάξης του 10^{-8} προς αποφυγή της διαίρεσης με το 0. Ένα κύριο όφελος του αλγορίθμου Adagrad είναι ότι εξαλείφει την ανάγκη για χειροκίνητη ρύθμιση του ρυθμού μάθησης, οι περισσότερες υλοποιήσεις θέτουν μια τιμή 0.01 και το αφήνουν να εξελιχθεί. Από την άλλη πλευρά, βέβαια, έχει ένα σημαντικό μειονέκτημα καθώς με το πέρασμα των βημάτων προστίθενται συνεχώς θετικοί όροι στον παρονομαστή με αποτέλεσμα ο ρυθμός μάθησης να μειώνεται διαρκώς. Αυτό μπορεί να οδηγήσει σε απειροστά μικρό ρυθμό μάθησης στο σημείο που η εκπαίδευση δεν μπορεί να προσφέρει πλέον καμία γνώση.

Μία επέκταση του αλγορίθμου Adagrad αποτελεί ο Αλγόριθμος Προσαρμοσμένης Διαφοράς Δέλτα (Adaptive Delta Algorithm - Adadelta) [30] ο οποίος και χρησιμοποιείται αργότερα στην εκπαίδευση των νευρωνικών δικτύων του πειραματικού μέρους. Τα μειονεκτήματα λοιπόν που παρουσιάζει ο αλγόριθμος Adagrad αντιμετωπίζονται στον Adadelta ως εξής: Αρχικά, οι ενημερώσεις του διανύσματος κλίσης βασίζονται πλέον όχι σε όλες τις προηγούμενες τιμές μέχρι την χρονική στιγμή t του βήματος αλλά περιορίζονται σε ένα παράθυρο w των τελευταίων τιμών. Με αυτόν τον τρόπο το άθροισμα στον παρονομαστή της εξίσωσης 2.9 δεν φτάνει σε πολύ μεγάλες τιμές και έτσι εξασφαλίζεται ότι επιτυγχάνεται συνεχής πρόοδος στην εκπαίδευση ακόμα και μετά από πολλά βήματα. Δεύτερον, στην μέθοδο αυτή αθροίζονται πλέον όχι τα τετράγωνα των διανυσμάτων κλίσης αλλά ένας εκθετικά μειούμενος μέσος όρος των τετραγώνων. Θεωρώντας ρ μια σταθερά μείωσης και γράφοντας ως g_i την κλίση στο χρονικό βήμα i, ο όρος αυτός φαίνεται στην εξίσωση 2.5:

$$E[g^2]_i = \rho E[g^2]_{i-1} + (1-\rho) g^2{}_i \quad (2.5)$$

Παίρνοντας τελικά την ρίζα αυτής της ποσότητας και προσθέτοντας πάλι έναν όρο εξομάλυνσης ε προκύπτει η Ρίζα Μέσου Τετραγωνικού Σφάλματος (Root Mean Square Error – RMS) όπως φαίνεται στην εξίσωση 2.6

$$RMS[g]_i = \sqrt{E[g^2]_i + \varepsilon} \quad (2.6)$$

Έτσι η εξίσωση 2.4 μετασχηματίζεται ως:

$$\vec{w}_{i+1} = \vec{w}_i - \frac{\eta}{RMS[g]_i} \cdot g_i \quad (2.7)$$

Προκειμένου, όμως, να αποφευχθεί το πρόβλημα ασυμφωνίας μονάδων στην εξίσωση 2.7 (παράμετροι – κλίση), ορίστηκε αντιστοίχως το $RMS[\Delta_w]_i$, δηλαδή η Ρίζα Μέσου Τετραγωνικού Σφάλματος των παραμέτρων. Αντικαθιστώντας τελικά τον ρυθμό μάθησης η με το $RMS[\Delta_w]_{i-1}$(καθώς δεν είναι γνωστό για το χρονικό βήμα i) έχουμε την μέθοδο Adadelta στην εξίσωση 2.8

$$\vec{w}_{i+1} = \vec{w}_i - \frac{RMS[\Delta_w]_{i-1}}{RMS[g]_i} \cdot g_i \quad (2.8)$$

Καταλήγουμε, λοιπόν, σε μία διαδικασία ανεξάρτητη του ρυθμού μάθησης καθώς έχει εξαλειφθεί από την ενημέρωση σε κάθε βήμα.

2.3.2 Βήματα Εκπαίδευσης Νευρωνικών Δικτύων

Έχοντας αναλύσει τις απαραίτητες έννοιες σχετικά με την δομή, την αρχιτεκτονική και τις μαθηματικές εκφράσεις, μπορούμε να περιγράψουμε συνοπτικά την συνολική διαδικασία που ακολουθείται στην εκπαίδευση νευρωνικών δικτύων. Ο αλγόριθμος εκπαίδευσης, λοιπόν, μπορεί να αναλυθεί συνοπτικά στα παρακάτω βήματα:

- Πέρασμα των δεδομένων εισόδου (συνήθως σε μικρό-ομάδες / batches) μέσα από το δίκτυο και εύρεση κατανομής εξόδου για κάθε στοιχείο.

- Υπολογισμός συνάρτησης κόστους ως προς την εκάστοτε πραγματική κατηγορία αντικειμένων.

- Πέρασμα προς τα πίσω από το τελευταίο επίπεδο προς το πρώτο μέσω του κανόνα αλυσίδας όπου εφαρμόζεται οπισθοδρόμηση προκειμένου να εφαρμοστεί ο εκάστοτε αλγόριθμος υπολογισμού κλίσης (στην περίπτωση της διπλωματικής εργασίας αυτής ο αλγόριθμος Adadelta) και να ενημερωθούν οι παράμετροι του δικτύου (βάρη κλπ).

Τα βήματα αυτά επαναλαμβάνονται όσες φορές επιλέξει ο χρήστης πάνω στα δεδομένα εισόδου και ονομάζονται εποχές. Η εκπαίδευση του δικτύου ολοκληρώνεται είτε όταν ολοκληρωθούν όλες οι εποχές είτε αν γίνει αληθής κάποια προκαθορισμένη συνθήκη τερματισμού (early stopping) που συνήθως σχετίζεται με κάποιο όριο στις τιμές της συνάρτησης κόστους [31].

2.4 Κατανεμημένη Εκπαίδευση Νευρωνικών Δικτύων

Σε αυτήν την ενότητα θα διερευνηθεί το κομμάτι της κατανεμημένης εκπαίδευσης νευρωνικών δικτύων [8], [9]. Με τον όλο και περισσότερο αυξανόμενο όγκο δεδομένων και τις μεγαλύτερες απαιτήσεις σε ακρίβεια και πολυπλοκότητα των δικτύων, γεννήθηκε η ανάγκη για αλγορίθμους βαθιάς μηχανικής μάθησης μεγάλης κλίμακας. Μία πρώτη κίνηση που έφερε πρόοδο ήταν η επιστράτευση συσκευών GPU για επιτάχυνση της διαδικασίας εκπαίδευσης. Παρ' όλα αυτά, η τεχνική αυτή από μόνη της έχει τον περιορισμό ότι η επιτάχυνση που επιτυγχάνεται είναι μικρή όταν το μοντέλο δεν χωράει στην μνήμη των GPU. Έτσι, πραγματοποιήθηκε μια διαφορετική προσέγγιση: η χρήση μεγάλης κλίμακας ομάδων μηχανών (large-scale clusters) με σκοπό την κατανομή και διαμοιρασμό του φόρτου τόσο σε επίπεδο μοντέλου όσο και σε επίπεδο δεδομένων.

2.4.1 Τεχνικές Παραλληλοποίησης

Πρώτον, ο λεγόμενος παραλληλισμός δεδομένων (data parallelism) απευθύνεται στην διαίρεση των δεδομένων εισόδου σε ίσα μέρη και τον διαμοιρασμό τους στους κόμβους του συστήματος. Με την σειρά τους, ο κάθε κόμβος του συστήματος έχει ένα αντίγραφο του νευρωνικού δικτύου με τα δικά του τοπικά βάρη κάθε φορά τα οποία ενημερώνει βάσει των δεδομένων που του

μοιράστηκαν και τέλος κάθε κόμβος κοινοποιεί τα τοπικά του βάρη και μέσω ενός αλγορίθμου συλλογής υπολογίζονται τα νέα καθολικά βάρη κ.ο.κ.

Δεύτερον, ο παραλληλισμός μοντέλου (model parallelism) επιχειρεί να κατανείμει την διαδικασία της εκπαίδευσης χωρίζοντας την αρχιτεκτονική του μοντέλου σε ξεχωριστούς κόμβους. Η τεχνική αυτή χρησιμοποιείται συνήθως όταν το μοντέλο είναι πολύ μεγάλο για να χωρέσει σε ένα μόνο μηχάνημα.

2.4.2 Θέματα Συγχρονισμού

Η διαδικασία εκπαίδευσης των νευρωνικών δικτύων βασίζεται κυρίως στην επαναληπτική ενημέρωση των παραμέτρων του μοντέλου μέσω αλγορίθμων όπως ο SGD που αναφέρθηκε νωρίτερα και άλλοι. Όταν πρόκειται, όμως, για κατανεμημένη εκπαίδευση, συνήθως αυτοί οι αλγόριθμοι χωρίζονται περαιτέρω σε δύο κατηγορίες, την σύγχρονη υλοποίηση και την ασύγχρονη.

Στην σύγχρονη υλοποίηση, οι κόμβοι του κατανεμημένου δικτύου είναι στενά συνδεδεμένοι. Κάθε κόμβος υπολογίζει τα διανύσματα κλίσης πάνω στο τοπικό κομμάτι των δεδομένων που του έχουν δοθεί. Στην συνέχεια όλες αυτές οι κλίσεις αποστέλλονται στον διακομιστή-αφέντη (master server) ο οποίος αναλαμβάνει την συλλογή τους και υπολογίζει τον μέσο όρο τους ώστε να υπολογιστούν οι νέες τιμές παραμέτρων που θα διαμοιραστούν στους κόμβους για να συνεχίσει η διαδικασία με το επόμενο βήμα. Αυτή η τεχνική είναι ανάλογη αυτής σε ένα μόνο μηχάνημα και ουσιαστικά εγγυάται την επιθυμητή σύγκλιση. Έχει, παρ' όλα αυτά, μερικά μειονεκτήματα:

- Σε ένα κατανεμημένο σύστημα, υπάρχει πιθανότητα κάποια μηχανήματα να αργούν να αποκριθούν λόγω πχ κακής ποιότητας δικτύου ή σφαλμάτων συστήματος. Αντίστοιχα, κάποιοι υπολογισμοί κλίσεων, ιδιαίτερα προς το τέλος της εκπαίδευσης, ενδέχεται να είναι αρκετά επιβαρυμένοι και εν τέλει χρονοβόροι. Και οι δύο αυτές περιπτώσεις προκαλούν καθυστέρηση σε όλη την διαδικασία καθώς όλο το σύστημα περιμένει αυτά τα μηχανήματα να αποκριθούν. Συχνά αυτό το πρόβλημα αναφέρεται ως Φράγμα Συγχρονισμού (Synchronization Barrier)

- Όλοι οι κόμβοι-εργάτες (worker nodes) επικοινωνούν με τον ίδιο master node αφήνοντας έτσι το σύστημα ευάλωτο σε αποτυχία από ένα συγκεκριμένο σημείο (single point of failure). Αυτό το ευάλωτο σημείο αποφεύγεται με την χρήση αλγορίθμων All Reduce που είναι τύπου peer-to-peer και αναλύεται στην επόμενη υπό-ενότητα.

- Παρόμοιο πρόβλημα υφίσταται και στο γεγονός ότι αυτή η τεχνική δεν προσφέρει ανοχή σε σφάλματα (fault tolerance) που σημαίνει ότι σε περίπτωση αποτυχίας πρέπει να επανεκκινηθεί η διαδικασία.

Στην ασύγχρονη υλοποίηση, πολλαπλά αντίγραφα του μοντέλου εκπαιδεύονται παράλληλα σε διαφορετικούς κόμβους με διαφορετικά υποσύνολα των δεδομένων. Κάθε αντίγραφο μοντέλου δέχεται τα καθολικά βάρη από τον διακομιστή παραμέτρων, υπολογίζει τα διανύσματα κλίσης πάνω στα δεδομένα του και επιστρέφει πίσω τις τιμές στον διακομιστή για να ενημερωθούν τα βάρη αναλόγως. Με αυτόν τον τρόπο, οι κόμβοι λειτουργούν ανεξάρτητα ο ένας με τον άλλον και έτσι το κάθε μηχάνημα εργάζεται στον ρυθμό του, δίνοντας μεγαλύτερη ευρωστία σε σφάλματα και ξεπερνώντας εν τέλει το φράγμα συγχρονισμού της σύγχρονης υλοποίησης. Η διαφορετική αυτή προσέγγιση, όμως, εμπεριέχει και κάποια μειονεκτήματα:

- Όπως και στην σύγχρονη υλοποίηση, έτσι και στην ασύγχρονη υπάρχει το ευάλωτο σημείο του single point failure καθώς και πάλι χρησιμοποιείται η δομή master-worker.

- Το βασικό, όμως, ζήτημα της ασύγχρονης υλοποίησης είναι τα «καθυστερημένα» διανύσματα κλίσης. Από την στιγμή που δεν υφίσταται συγχρονισμός μεταξύ κόμβων, κάποιοι workers ενδέχεται να κάνουν υπολογισμούς πάνω σε τιμές παραμέτρων που είναι αρκετά βήματα πίσω από την τρέχουσα έκδοσή τους. Το αποτέλεσμα είναι η επιθυμητή σύγκλιση να καθυστερεί και επίσης να μην είναι εγγυημένη. Διάφοροι τρόποι έχουν προταθεί για να επιτευχθεί εγγυημένα η σύγκλιση στην ασύγχρονη υλοποίηση όπως η μέθοδος εναλλασσόμενης κατεύθυνσης πολλαπλασιαστών (alternating direction method of multipliers - ADMM algorithm) [32]

Τα στοιχεία αυτά γύρω από το ζήτημα συγχρονισμού μπορούν να συνοψιστούν στον Πίνακα 2.1(με μαύρο φαίνονται ουδέτερα, με κόκκινο αρνητικά και με πράσινο θετικά χαρακτηριστικά).

Πίνακας 2.1: Χαρακτηριστικά σύγχρονης και ασύγχρονης εκπαίδευσης

Χαρακτηριστικό	**Σύγχρονη Εκπαίδευση**	**Ασύγχρονη Εκπαίδευση**
Στενά συνδεδεμένοι κόμβοι	✓	
Αντίγραφα μοντέλου		✓
Ανοχή σε σφάλματα		✓
Single point failure	✓	✓
Φράγμα Συγχρονισμού	✓	
Καθυστερημένη / Μη-εγγυημένη σύγκλιση		✓

2.4.3 Συνήθεις Κατανεμημένες Αρχιτεκτονικές Εκπαίδευσης Νευρωνικών Δικτύων

Δύο συνήθεις αρχιτεκτονικές που απευθύνονται σε κατανεμημένη εκπαίδευση νευρωνικών δικτύων είναι η αρχιτεκτονική *Διακομιστή Παραμέτρων* και η τεχνική *AllReduce* και αναλύονται στις επόμενες υπό-ενότητες.

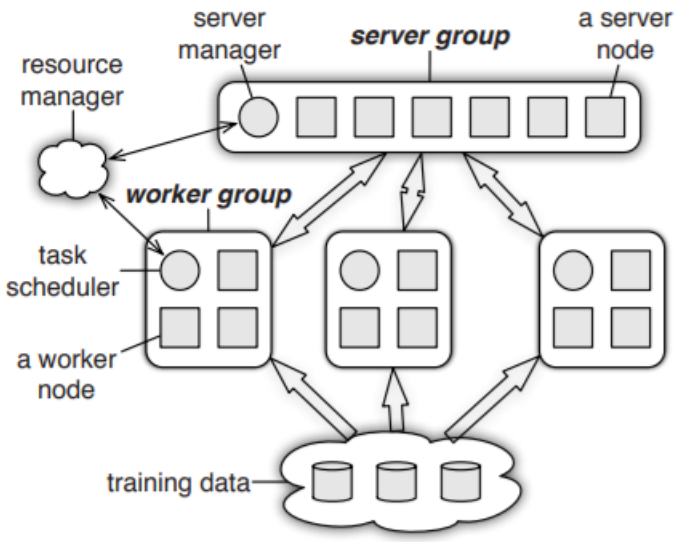
2.4.3.1 Διακομιστής Παραμέτρων (Parameter Server)

Μια σημαντική αρχιτεκτονική που απευθύνεται σε προβλήματα κατανεμημένης εκπαίδευσης νευρωνικών δικτύων είναι η αρχιτεκτονική που χρησιμοποιεί τον λεγόμενο Διακομιστή Παραμέτρων (Parameter Server) [7], [33]. Σε αυτήν την αρχιτεκτονική, οι κόμβοι χωρίζονται μία ομάδα διακομιστών (server group) και πολλές ομάδες εργατών (worker groups).

Στην ομάδα διακομιστή, οι κόμβοι διακομιστές διατηρούν ένα μέρος των καθολικά διαμοιρασμένων παραμέτρων και επικοινωνούν μεταξύ τους για να αντιγράψουν ή/και να ενσωματώσουν παραμέτρους. Στην ομάδα αυτή συνήθως υπάρχει και ένας κόμβος διαχειριστής (manager node) που διατηρεί μετά-δεδομένα του δικτύου όπως κατάσταση κόμβων και ανάθεση παραμέτρων.

Όσον αφορά τις ομάδες εργατών, κάθε μία εκτελεί μία διεργασία, κρατώντας τοπικά ένα μέρος των δεδομένων εκπαίδευσης, για παράδειγμα τον υπολογισμό διανυσμάτων κλίσης. Οι κόμβοι

εργάτες επικοινωνούν μόνο με τους διακομιστές και όχι μεταξύ τους, μεταδίδοντας τις ενημερωμένες τιμές τους και λαμβάνοντας τις διαμοιρασμένες παραμέτρους. Σε κάθε ομάδα υπάρχει και ένας κόμβος υπεύθυνος προγράμματος (scheduler node) του οποίου δουλειά είναι να αναθέτει εργασίες στους κόμβους εργάτες και να επιτηρεί την πρόοδο των υπολογισμών. Οι διεργασίες που εκτελούνται σε κάθε ομάδα εργατών είναι ανεξάρτητες και επίσης μπορούν πολλές τέτοιες ομάδες να ανατίθενται στο ίδιο πρόβλημα, επιτυγχάνοντας έτσι τον απαραίτητο παραλληλισμό και κατανέμοντας την εκπαίδευση νευρωνικών δικτύων. Στην Εικόνα 2.6 [7] παρατίθεται η αρχιτεκτονική που εφαρμόζεται με την χρήση διακομιστή παραμέτρων.



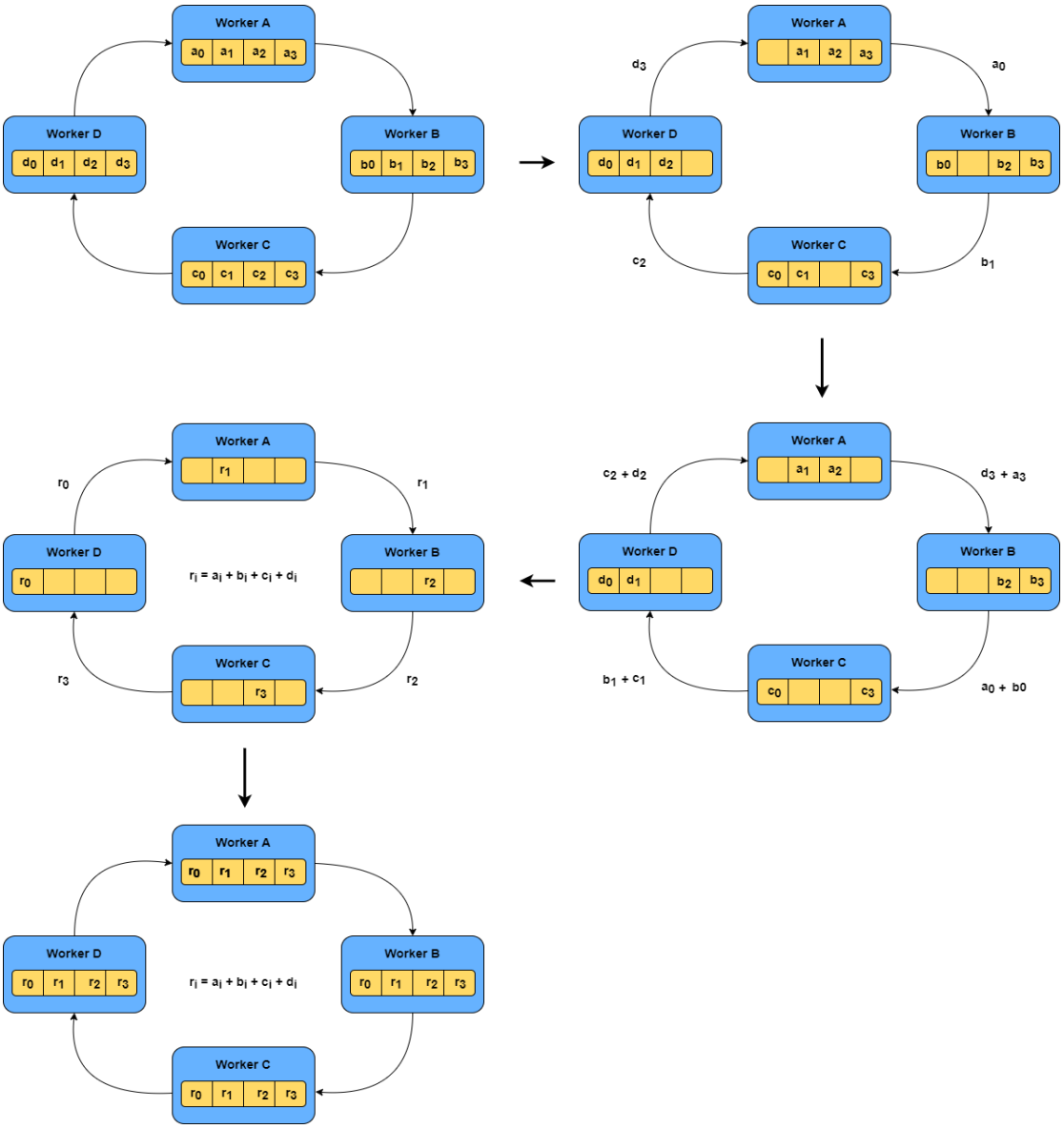
Εικόνα 2.6: Αρχιτεκτονική Διακομιστή Παραμέτρων

2.4.3.2 Τεχνική AllReduce

Το κομμάτι της επικοινωνίας μεταξύ κόμβων και της συλλογής επιμέρους διανυσμάτων κλίσεων είναι πολύ βασικό όταν πραγματοποιείται κατανεμημένη εκπαίδευση. Σε προηγούμενη υπό-ενότητα αναλύθηκε η αρχιτεκτονική με διακομιστή παραμέτρων πάνω στην λογική master-worker.

Μία διαφορετική προσέγγιση της επικοινωνίας είναι η ανταλλαγή δεδομένων της μορφής peer-to-peer, δηλαδή σε ζευγάρια. Αυτής της φιλοσοφίας είναι οι αλγόριθμοι All Reduce [9]. Αυτό που κάνουν οι αλγόριθμοι αυτού του είδους είναι να εκτελούν μια πράξη/διεργασία στην συλλογή των δεδομένων από κάθε κόμβο-μηχάνημα και να διαμοιράζουν εκ νέου το αποτέλεσμα αυτό. Οι αλγόριθμοι All Reduce χρησιμοποιούνται ευρέως στην σύγχρονη εκπαίδευση νευρωνικών δικτύων. Από τις πιο διαδεδομένες εκδόσεις η οποία χρησιμοποιείται και αργότερα στο πειραματικό μέρος της διπλωματικής ονομάζεται Ring All Reduce [34] και περιγράφεται ακολούθως.

Ο αλγόριθμος Ring All Reduce χωρίζεται σε δύο επιμέρους διαδικασίες. Η πρώτη διαδικασία ονομάζεται scatter-reduce. Σε αυτήν την φάση, τα δεδομένα διαιρούνται με τον αριθμό των κόμβων του δικτύου, έστω n, και κάθε κόμβος i στέλνει δεδομένα στον κόμβο (i+1) % n όπου % είναι ο τελεστής υπολοίπου modulo. Έπειτα, κάθε κόμβος εφαρμόζει την πράξη/διεργασία στο κομμάτι δεδομένων που έχει και το στέλνει εκ νέου στον επόμενο κόμβο για να συνεχιστεί η διαδικασία. Αυτή η αλληλουχία γίνεται n-1 φορές. Όταν τελειώσει αυτή η φάση, το μόνο που μένει είναι να μοιραστεί κάθε κόμβος το μέρος του αποτελέσματος που έχει με όλους τους υπόλοιπους μέσω μετάδοσης. Αυτή η διαδικασία ονομάζεται all-gather, γίνεται επίσης n-1 φορές και ακολουθεί το μοτίβο της πρώτης, με μόνη διαφορά ότι απλά διαμοιράζονται και αποθηκεύονται δεδομένα χωρίς να εκτελείται κάποια πράξη. Στην Εικόνα 2.7 φαίνεται μια σχηματική αναπαράσταση του αλγορίθμου αυτού.



Εικόνα 2.7: Βήματα Αλγορίθμου Ring All Reduce

Τα πλεονεκτήματα και μειονεκτήματα αυτής της προσέγγισης μπορούν συνοπτικά να παρατεθούν ως εξής:

+ Αποτελεσματική χρήση του εύρους ζώνης του δικτύου καθώς κανένα μηχάνημα δεν παραμένει ανενεργό περιμένοντας
+ Αποφυγή του single point of failure εφόσον χρησιμοποιείται επικοινωνία peer-to-peer και όχι master-worker
+ Ανεξαρτησία από τον αριθμό και την τοπολογία των μηχανών

– Τα βήματα που χρειάζονται για κάθε μία από τις δύο φάσεις είναι n-1. Άρα έχουμε πολυπλοκότητα της τάξης O(n) ενώ άλλοι αλγόριθμοι All Reduce επιτυγχάνουν O(logn)
– Δεν προσφέρεται ανοχή σε σφάλματα. Αν κάποιο μηχάνημα αποτύχει, η διαδικασία πρέπει να ξεκινήσει εκ νέου από την αρχή.

Κεφάλαιο 3

Αρχιτεκτονικές Συστημάτων

Στο κεφάλαιο αυτό θα γίνει μία αναλυτική περιγραφή της αρχιτεκτονικής, των δομικών στοιχείων και του τρόπου λειτουργίας των συστημάτων που θα χρησιμοποιηθούν αργότερα στο πειραματικό μέρος της διπλωματικής εργασίας. Στην πρώτη ενότητα θα γίνει βιβλιογραφική ανάλυση του συστήματος TensorFlow, στην επόμενη του συστήματος Spark και στην τελευταία του λογισμικού Horovod, το οποίο χρησιμεύει στην σύνδεση και επικοινωνία των δύο προαναφερθέντων.

3.1 Σύστημα TensorFlow

Το σύστημα TensorFlow [4], [35] αποσκοπεί στην υλοποίηση και ανάπτυξη μοντέλων μηχανικής και βαθιάς μάθησης μεγάλης κλίμακας, όπως για παράδειγμα αναγνώριση εικόνων, κατηγοριοποίηση χειρόγραφων ψηφίων, επεξεργασία φυσικής γλώσσας και πολλά άλλα. Η διεπαφή του χρήστη με το σύστημα γίνεται κυρίως με τις γλώσσες προγραμματισμού C++ και Python.

3.1.1 Αρχιτεκτονική και Λειτουργικότητα

Βασικές Έννοιες

Ένα υπολογιστικό φορτίο στο TensorFlow παριστάνεται από έναν *κατευθυνόμενο γράφο* με την έννοια *ροής δεδομένων* (*dataflow*). Κάθε κόμβος έχει μηδέν η περισσότερες εισόδους και μηδέν ή περισσότερες εξόδους και αντιπροσωπεύει την αρχικοποίηση μιας *διεργασίας* (*operation*). Οι τιμές που μεταφέρονται μέσω των ακμών του γράφου ονομάζονται *τανυστές (tensors)*. Ο τανυστής ορίζεται σαν πολυδιάστατος πίνακας και έχει τα εξής χαρακτηριστικά:

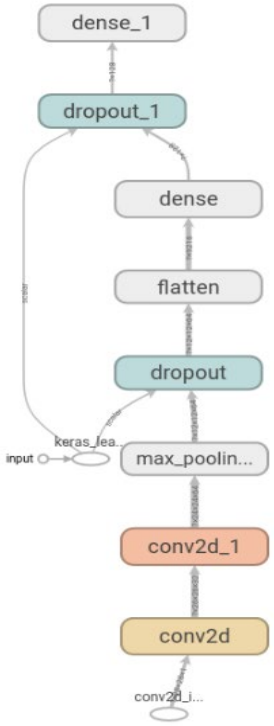
- **τύπο δεδομένων**: κάθε στοιχείο του τανυστή πρέπει να είναι του ίδιου τύπου δεδομένων όπως ακέραιος (integer), συμβολοσειρά (string) κλπ
- **μέγεθος κάθε διάστασης (shape)**
- **βαθμίδα (rank/order):** αριθμός διαστάσεων, αναφορικά βάση της βαθμίδας ένας τανυστής μπορεί να ονομάζεται *βαθμωτος (scalar)* με βαθμίδα 0, *διάνυσμα (vector)* με βαθμίδα 1, *πίνακας (matrix)* με βαθμίδα 2 και από κει και έπειτα *ν-τανυστής (n-Tensor)*.

Μια διεργασία (operation), που αναφέρθηκε νωρίτερα, αντιπροσωπεύει έναν γενικό υπολογισμό όπως πρόσθεση, πολλαπλασιασμό πινάκων κλπ. Μέσω διαφόρων ιδιοτήτων οι διεργασίες μπορούν να είναι *πολυμορφικές* και να λειτουργούν με διαφορετικό τύπο δεδομένων κάθε φορά. Ένα ιδιαίτερο και βασικό είδος διεργασίας ονομάζεται *μεταβλητή (variable)* και κρατάει έναν σύνδεσμο σε έναν σταθερά αποθηκευμένο (persistent) τανυστή που έχει πάντα κάποια αρχική τιμή και μπορεί να διαφοροποιείται και να επιβιώνει σε διάφορες εκτελέσεις ενός γράφου. Στις εφαρμογές μηχανικής μάθησης, οι παράμετροι του μοντέλου συνήθως διατηρούνται σε τανυστές που αποθηκεύονται σε μεταβλητές και ενημερώνονται κατά την διάρκεια της εκπαίδευσης. Αντίστοιχα

υπάρχουν και οι *αντικαταστάτες (placeholders)* που μοιάζουν με τις μεταβλητές αλλά δεν έχουν απαραίτητα κάποια αρχική τιμή. Χρησιμοποιούνται κυρίως για παροχή δεδομένων στον γράφο εκτέλεσης. Οι διεργασίες υλοποιούνται με συγκεκριμένο τρόπο σε *πυρήνες (kernels)* που μπορούν να εκτελεστούν σε έναν τύπο συσκευής όπως CPU, GPU ή και TPU.

Οι χρήστες αλληλεπιδρούν με το σύστημα του TensorFlow μέσω της διεπαφής *Session*. Βασική λειτουργία της διεπαφής αυτής είναι η *Εκτέλεση (Run)* που δέχεται τις εξόδους που πρέπει να υπολογιστούν και προαιρετικά μια ομάδα τανυστών ως παροχή δεδομένων σε κάποιους κόμβους του γράφου. Το σύστημα μετά αναλαμβάνει να υπολογίσει την μεταβατική κλειστότητα των κόμβων που πρέπει να εκτελεστούν για να παραχθεί το επιθυμητό αποτέλεσμα και στην συνέχεια να προχωρήσει στην εκτέλεσής τους με σειρά που σέβεται τις μεταξύ τους εξαρτήσεις.

Ένα παράδειγμα ενός γράφου εκτέλεσης στο σύστημα TensorFlow, συγκεκριμένα από το πρώτο νευρωνικό δίκτυο του πειραματικού μέρους, φαίνεται παρακάτω στην Εικόνα 3.1



Εικόνα 3.1: Παράδειγμα γράφου εκτέλεσης στο TensorFlow

Δομικά Στοιχεία

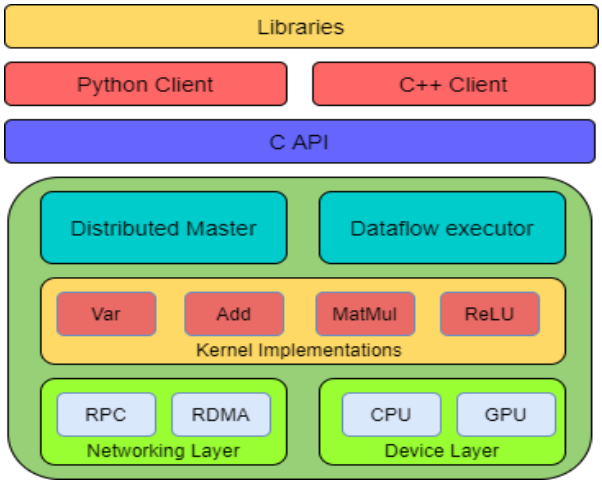
Τα κύρια δομικά στοιχεία του συστήματος είναι:

- το *πρόγραμμα-πελάτης* (*client)*, στον οποίο γράφει ο χρήστης το πρόγραμμα με τις διαθέσιμες βιβλιοθήκες και Διεπαφές Προγραμματισμού Εφαρμογών

- ο *αφέντης* (*master),* με τον οποίο επικοινωνεί ο client μέσω της διεπαφής Session. Κύρια αρμοδιότητα του master είναι η μετάφραση των αιτημάτων χρήστη σε πλάνο εκτέλεσης πάνω σε ομάδα *εργασιών (tasks)*. Έχοντας ως δεδομένο έναν γράφο, περικόπτει και διαμοιράζει υπο-γράφους στις διαθέσιμες συσκευές και τους αποθηκεύει στην κρυφή μνήμη (cache) για να επαναχρησιμοποιηθούν σε επόμενα βήματα της εκτέλεσης, εφαρμόζοντας παράλληλα βελτιστοποιήσεις όπως εξάλειψη περιττών υπο-εκφράσεων.

- οι *εργάτες (workers)*, οι οποίοι αναλαμβάνουν την εκτέλεση των κόμβων των υπό-γράφων που τους δίνονται, σύμφωνα με τις οδηγίες του master, πάνω σε πυρήνες που ενεργούν στις διαθέσιμες υπολογιστικές συσκευές (CPU, GPU κλπ).

Μία εικόνα των επιμέρους δομικών στοιχείων του συστήματος φαίνεται στην Εικόνα 3.2



Εικόνα 3.2: Δομικά στοιχεία συστήματος TensorFlow

Διακομιστές Παραμέτρων

Σημαντικό κομμάτι του συστήματος TensorFlow είναι ο *διακομιστής παραμέτρων (parameter server)*. Στην αρχιτεκτονική αυτή, όπως αναφέρθηκε και στην ανάλυση προηγούμενης ενότητας, μία εργασία συγκροτείται από δύο ανεξάρτητες ομάδες διαδικασιών: διαδικασίες εργάτη χωρίς αναφορά κατάστασης (stateless worker processes) που εκτελούν το μεγαλύτερο μέρος των απαραίτητων υπολογισμών στην εκπαίδευση του μοντέλου και διαδικασίες διακομιστή παραμέτρων με αναφορά κατάστασης (stateful parameter server processes) που διατηρούν την τρέχουσα έκδοση των παραμέτρων του μοντέλου.

Στην πράξη, το TensorFlow δεν χρησιμοποιεί επακριβώς διακομιστές παραμέτρων. Σε ένα σύμπλεγμα μηχανών, αναπτύσσεται ένα σετ εργασιών που επικοινωνούν μέσω ενός δικτύου και κάνουν χρήση της ίδιας διεπαφής για εκτέλεση γράφων. Τυπικά, κάποιες από αυτές τις εργασίες αναλαμβάνουν τον ρόλο που παίζει ένας διακομιστής παραμέτρων αλλά έχουν την δυνατότητα να εκτελούν πολλές διαφορετικές λειτουργικότητες και είναι πιο ευέλικτοι από τους συμβατικούς διακομιστές παραμέτρων.

Χαρακτηριστικά – Πλεονεκτήματα

Ένα χαρακτηριστικό των εφαρμογών σε TensorFlow είναι η εκτέλεση σε δύο διακριτές φάσης. Στην πρώτη φάση ορίζεται το πρόγραμμα (το νευρωνικό δίκτυο κλπ) ως ένας συμβολικός γράφος ροής δεδομένων με αντικαταστάτες και μεταβλητές. Στην δεύτερη φάση εκτελείται μια βελτιστοποιημένη έκδοση του προγράμματος στις διαθέσιμες συσκευές, δίνοντας έτσι την δυνατότητα βελτιστοποίησης της εκτέλεσης χρησιμοποιώντας καθολικές πληροφορίες για όλο τον υπολογισμό.

Ένα σημαντικό γνώρισμα του TensorFlow είναι η υποστήριξη για αυτόματο υπολογισμό κλίσης (gradient), που είναι βασικό κομμάτι στις συναρτήσεις κόστους πολλών αλγορίθμων βελτιστοποίησης. Εάν μέσω του γράφου ένας τανυστής έχει εξάρτηση από μια ομάδα άλλων τανυστών, το σύστημα προσφέρει ενσωματωμένες συναρτήσεις για υπολογισμό των κλίσεων αυτών.

Η διαδικασία αυτή ξεκινάει από τον εντοπισμό του μονοπατιού στον γράφο που οδηγεί στον υπολογισμό του τανυστή αναφοράς και έπειτα το σύστημα ανατρέχει (backtrack) προς τα πίσω και προσθέτει έναν κόμβο στον γράφο για κάθε διεργασία που γίνεται στο συγκεκριμένο μονοπάτι.

Διαφορετικές Δυνατότητες Υλοποίησης

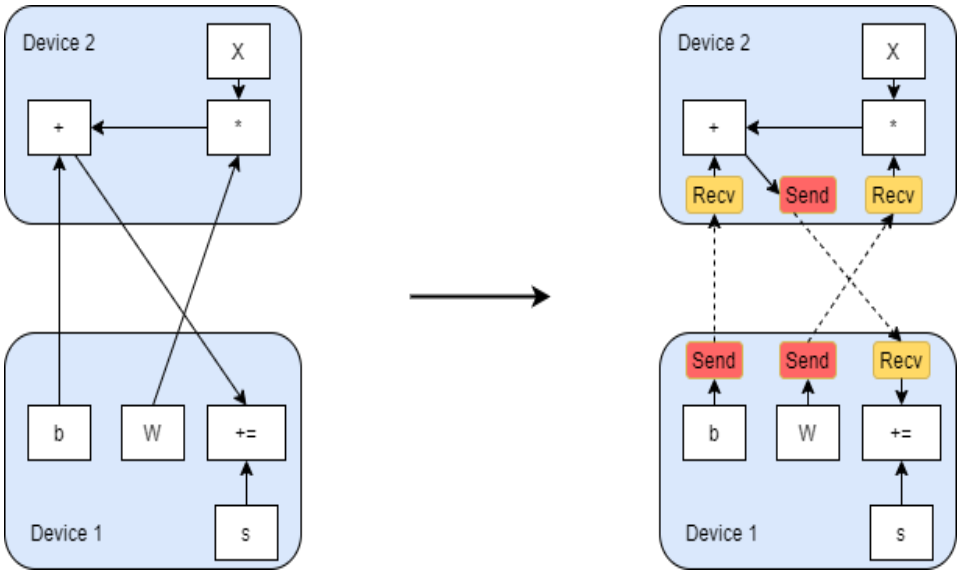
Το TensorFlow προσφέρει τόσο τοπικές (local) υλοποιήσεις όσο και κατανεμημένες (distributed). Η τοπική υλοποίηση χρησιμοποιείται όταν ο client, ο master και οι workers όλοι τρέχουν σε μία μηχανή, πιθανώς με πολλαπλές συσκευές αν για παράδειγμα υπάρχουν αρκετές GPU εγκατεστημένες. Η κατανεμημένη υλοποίηση μοιράζεται το μεγαλύτερο μέρος κώδικα με την τοπική αλλά την επεκτείνει παρέχοντας περιβάλλον στο οποίο οι διεργασίες των client, master και workers μπορούν να τρέχουν σε διαφορετικές μηχανές. Οι υλοποιήσεις παρατίθενται αναλυτικότερα παρακάτω.

Η εκτέλεση σε *μία μοναδική συσκευή (single-device execution)* έχει μία μοναδική διεργασία εργάτη. Οι κόμβοι του γράφου πρέπει να εκτελεστούν με σεβασμό στις αλληλεξαρτήσεις τους. Συγκεκριμένα, το σύστημα κρατάει για κάθε κόμβο των αριθμό εξαρτήσεων που έχει προκειμένου να εκτελεστεί. Όταν αυτός ο αριθμός γίνει μηδέν, ο κόμβος μπαίνει σε μια ουρά εκτέλεσης που λειτουργεί με ακαθόριστη σειρά. Όταν ένας κόμβος εκτελεστεί, τότε όλοι οι κόμβοι που εξαρτώνται από αυτόν μειώνουν τον αριθμό των εξαρτήσεων τους.

Στην υλοποίηση με *πολλαπλές συσκευές (multi-device execution)* το σύστημα καλείται να ορίσει ποια συσκευή θα αναλάβει τον υπολογισμό ποιων κόμβων και έπειτα να διευθύνει την απαραίτητη επικοινωνία μεταξύ τους.

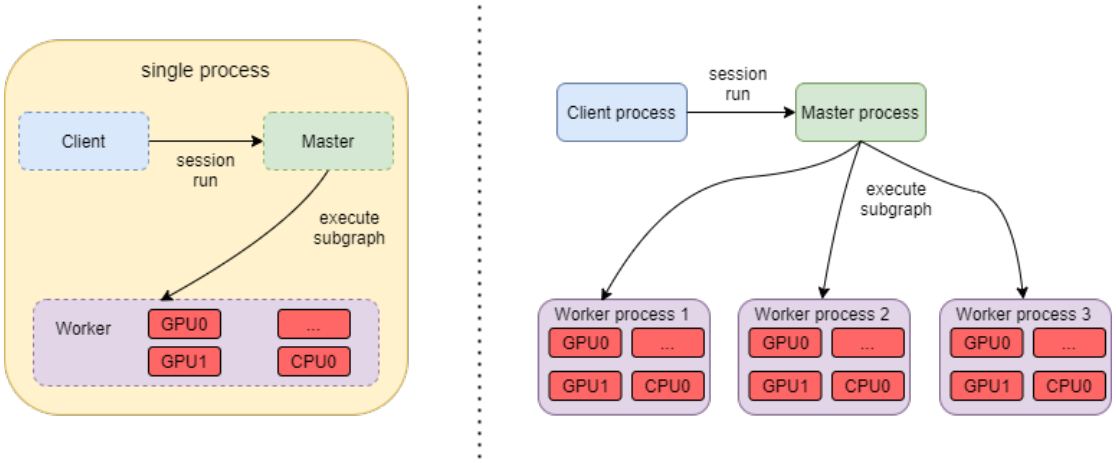
Για το πρώτο πρόβλημα το σύστημα χρησιμοποιεί έναν *αλγόριθμο τοποθέτησης (placement algorithm)*. Ειδικότερα, πρόκειται για ένα μοντέλο που περιέχει εκτιμήσεις για το μέγεθος των εισόδων και εξόδων κάθε κόμβου και για τον χρόνο υπολογισμού του. Δουλειά του μοντέλου είναι να τρέξει μια εικονική εκτέλεση του γράφου και να επιλέξει συσκευές για τους κόμβους χρησιμοποιώντας άπληστες ευρετικές (greedy heuristics).

Όταν τελειώσει ο αλγόριθμος τοποθέτησης κόμβων, κάθε συσκευή έχει αναλάβει έναν υπο-γράφο. Αυτό οδηγεί σε ακμές του συνολικού γράφου που επικοινωνούν από συσκευή σε συσκευή (cross-device communication). Αυτές οι ακμές αφαιρούνται και στην θέση τους μπαίνουν ένας κόμβος *Send* κι ένας *Recv* στους αντίστοιχους υπο-γράφους, απομονώνοντας έτσι την λειτουργία επικοινωνίας από την υπόλοιπη υλοποίηση. Οι εκτελέσεις των εργασιών Send και Recv μπορούν να χρησιμοποιούν διαφορετικές διεπαφές και πρωτόκολλα ανάλογα το είδος συσκευών που επικοινωνούν προκειμένου να επιτευχθεί μεγαλύτερη αποδοτικότητα. Στην Εικόνα 3.3 φαίνεται ένα παράδειγμα αυτής της επικοινωνίας.



Εικόνα 3.3: Επικοινωνία μεταξύ συσκευών

Τέλος, η κατανεμημένη εκτέλεση μοιάζει πολύ με την εκτέλεση με πολλαπλές συσκευές, διαμοιράζοντας πλέον τον φόρτο σε ένα σύμπλεγμα (cluster) μηχανών που μπορούν να έχουν πολλές συσκευές για εκτέλεση η καθεμία. Η ανοχή σε μερικές αποτυχίες είναι περιορισμένη καθώς όταν εντοπισθεί κάποιο σφάλμα εκτέλεσης, ολόκληρη η εκτέλεση του γράφου ματαιώνεται και επανεκκινείται από το μηδέν. Για αυτό το λόγο το σύστημα υποστηρίζει διαρκή *ενημέρωση σημείων ελέγχου (checkpointing)* και επανάκτηση εκείνης της κατάστασης στην επανεκκίνηση. Στην Εικόνα 3.4 γίνεται μια σχηματική αναπαράσταση της σύνδεσης των επιμέρους δομικών στοιχείων σε τοπική και κατανεμημένη υλοποίηση αντίστοιχα.



(α) Τοπική υλοποίηση (β) Κατανεμημένη υλοποίηση

Εικόνα 3.4: Σύνδεση δομικών στοιχείων για τοπική και κατανεμημένη υλοποίηση

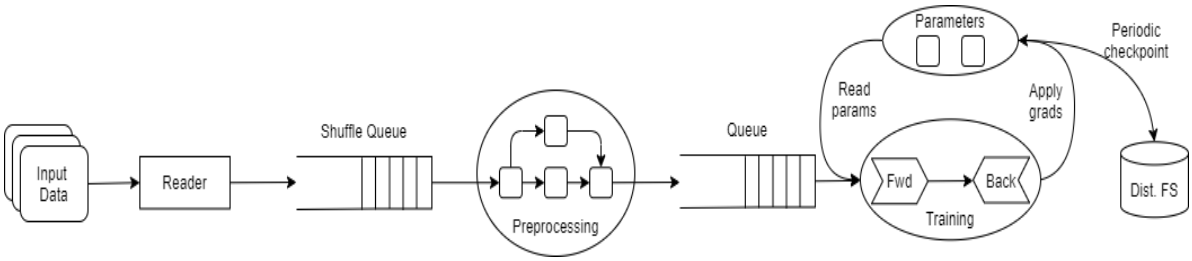
Δεύτερη Έκδοση – Eager Execution

Στην δεύτερη έκδοση του συστήματος TensorFlow υλοποιήθηκαν σημαντικές αλλαγές στην εκτέλεση του προγράμματος, στην αρχιτεκτονική και στην διεπαφή που προσφέρεται στον χρήστη προκειμένου να είναι πιο ευέλικτη, κατανοητή και βοηθητική σε ελέγχους (testing) και αποσφαλμάτωση (debugging).

Η πιο καίρια προσθήκη ήταν η λεγόμενη «άμεση εκτέλεση» *(eager execution)* [36], [37]. Σε αυτόν τον τρόπο λειτουργίας, οι χρήστες δεν απαιτείται να συντάξουν έναν γράφο εκτέλεσης ο οποίος θα υπολογιστεί στο τέλος μέσω της διεπαφής Session. Αντ' αυτού, κάθε διεργασία αποτιμάται άμεσα επιστρέφοντας απτές τιμές χωρίς να κατασκευάζεται κάποιος γράφος. Έτσι, κάθε τανυστής, πχ μια μεταβλητή, διατηρεί σταθερές τιμές δεδομένων και όχι μία αναφορά σε κάποιον κόμβο του γράφου (variable – operation). Εναλλακτική της διεπαφής Session για τους χρήστες αποτελεί η διεπαφή Function (tf.function) με την οποία το σύστημα μπορεί να εκτελέσει συναρτήσεις σε κώδικα (πχ Python) με την μορφή γράφου (ξεχωριστός γράφος για κάθε συνάρτηση που εκτελείται μέσω αυτής της διεπαφής) προκειμένου να επιτευχθούν αποδοτικότητα, βελτιστοποίηση και επαναχρησιμοποίηση.

Σωλήνωση Εισόδου – Παράδειγμα Ροής Εκπαίδευσης

Τέλος, έχοντας αναλύσει τις απαραίτητες έννοιες και τον τρόπο λειτουργίας του συστήματος, μπορούμε στην Εικόνα 3.5 να δούμε μια τυπική αλληλουχία διεργασιών που εκτελούνται κατά την διαδικασία εκπαίδευσης νευρωνικών δικτύων, όπως περιγράφηκε σε προηγούμενο κεφάλαιο, με την χρήση του συστήματος TensorFlow [38], [39].



Εικόνα 3.5: Παράδειγμα βημάτων εκπαίδευσης νευρωνικών δικτύων μέσω TensorFlow

Στην αρχή της ροής το σύστημα διαβάζει τα δεδομένα εισόδου από το σημείο αποθήκευσής τους και τα φέρνει στην μορφή που αναγνωρίζει για να τα επεξεργαστεί. Είναι σύνηθες μετά τα δεδομένα να ανακατεύονται (shuffle) και να αλλάζουν σειρά το οποίο εξυπηρετεί στο να μην αναγνωρίζει μοτίβα το σύστημα που εξαρτώνται από την σειρά των δεδομένων και να επιτυγχάνεται μεγαλύτερη γενίκευση του μοντέλου. Έπειτα, είναι απαραίτητο τα δεδομένα να έρθουν σε μορφή κατάλληλη προς τροφοδοσία στο νευρωνικό δίκτυο (preprocessing), συνήθως σε γραμμές όπου κάθε γραμμή είναι της μορφής [χαρακτηριστικά, ετικέτα αντικειμένου]. Αυτό τον σκοπό εξυπηρετούν συναρτήσεις βιβλιοθήκης της Python ή C++, όπως επίσης και μέθοδοι του TensorFlow όπως η *map*. Επόμενο μέρος της σωλήνωσης εισόδου αποτελεί η ομαδοποίηση (batching) των δεδομένων. Κάθε τέτοια μικρό-ομάδα αποτελεί ένα βήμα της κάθε εποχής εκπαίδευσης. Ένας πολύ χρήσιμος μετασχηματισμός στην αλληλουχία είναι η αποθήκευση των δεδομένων στην κρυφή μνήμη *cache* (είτε στην μνήμη RAM είτε σε τοπική αποθήκευση). Την πρώτη φορά που θα προσπελαστούν τα δεδομένα θα κρατηθούν στην cache και έτσι στις επόμενες επαναλήψεις θα αποφευχθούν περιττές διεργασίες όπως άνοιγμα αρχείων και διάβασμα δεδομένων εξοικονομώντας έτσι σημαντικό μέρος χρόνου από την εκπαίδευση. Εξίσου σημαντικό ρόλο παίζει και μία δυνατότητα που προσφέρει η βιβλιοθήκη του TensorFlow που ονομάζεται *prefetch*. Το σύστημα χρησιμοποιεί ένα ξεχωριστό νήμα για να παράγει τα δεδομένα του επόμενου βήματος πριν ολοκληρωθεί το τρέχον βήμα

καταφέρνοντας να απεμπλακεί ο χρόνος παραγωγής δεδομένων και κατανάλωσης τους και να μειωθεί αισθητά ο συνολικός χρόνος εκπαίδευσης.

Ολοκληρώνοντας, έτσι, την ροή της σωλήνωσης εισόδου, τροφοδοτείται το νευρωνικό δίκτυο με δεδομένα και εκτελείται η εκπαίδευση σε μικρό-ομάδες, εφαρμόζοντας κιόλας περιοδικά checkpointing για επαναφορά της τρέχουσας κατάστασης σε περίπτωση σφάλματος. Στην επόμενη υπό-ενότητα θα αναλυθεί η διαδικασία της κατανεμημένης εκπαίδευσης μέσω TensorFlow, οι ρυθμίσεις που γίνονται εσωτερικά και οι τροποποιήσεις στην αλληλουχία που περιγράφηκε.

3.1.2 Κατανεμημένη Εκπαίδευση μέσω TensorFlow

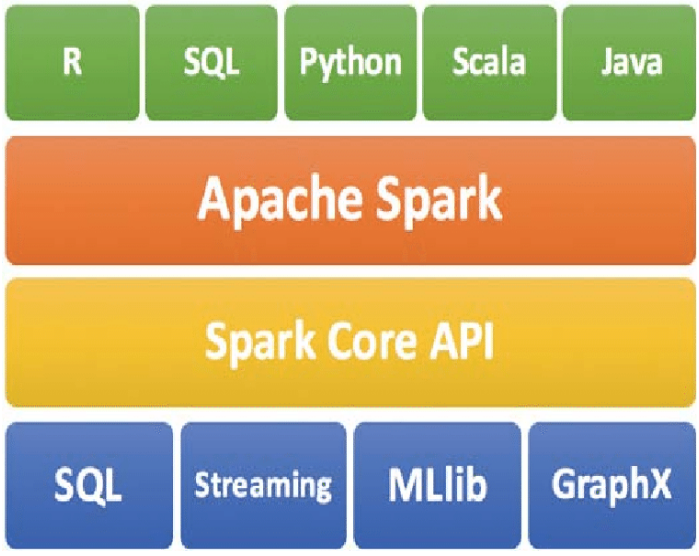
Στη παρούσα υπό-ενότητα θα γίνει μια σύντομη περιγραφή της κατανεμημένης υλοποίησης για εκπαίδευση νευρωνικών δικτύων που προσφέρει το TensorFlow [40], [41], δίνοντας βαρύτητα στην συγκεκριμένη μέθοδο που χρησιμοποιείται στο πειραματικό μέρος.

Το σύστημα προσφέρει την δυνατότητα να κατανεμηθεί ο φόρτος της εκπαίδευσης πάνω σε πολλαπλές συσκευές GPU ή και σε πολλαπλά μηχανήματα σε συμπλέγματα (clusters) είτε για σύγχρονη είτε για ασύγχρονη εκπαίδευση. Μία εκ των υλοποιήσεων εφαρμόζεται πάνω στην αρχιτεκτονική των parameter servers αλλά δεν θα γίνει περαιτέρω ανάλυση. Αυτή που χρησιμοποιείται και στο πειραματικό μέρος ονομάζεται *Στρατηγική Αντικατοπτρισμού σε Πολλαπλούς Εργάτες (MultiWorkerMirroredStrategy)*. Βάση αυτής αποτελεί η απλή Στρατηγική Αντικατοπτρισμού (MirroredStrategy) αλλά με επέκταση για πολλαπλά μηχανήματα. Η Στρατηγική Αντικατοπτρισμού προσφέρεται για σύγχρονη εκπαίδευση πάνω σε πολλαπλές GPU. Ουσιαστικά δημιουργεί ένα αντίγραφο του μοντέλου ανά συσκευή GPU, όπως επίσης και των μεταβλητών (Καθρεπτισμένες Μεταβλητές – MirroredVariables) που παραμένουν συγχρονισμένες εφαρμόζοντας πάνω τους πανομοιότυπες ενημερώσεις. Στην Στρατηγική Αντικατοπτρισμού σε Πολλαπλούς Εργάτες, λοιπόν, εφαρμόζεται αυτή η τεχνική σε πολλαπλά μηχανήματα, χρησιμοποιώντας για επικοινωνία (στην περίπτωσή μας) την υλοποίηση Ring AllReduce, που αναλύθηκε σε προηγούμενο κεφάλαιο, πάνω σε Απομακρυσμένες Κλήσεις Διαδικασιών (Remote Procedure Calls – RPC) [42] της Google.

Μία από τις απαραίτητες ρυθμίσεις για αυτή την υλοποίηση είναι η διαμόρφωση της δομής του cluster. Αυτό επιτυγχάνεται μέσω μιας μεταβλητής συστήματος που ονομάζεται "TF_CONFIG" στην οποία οι χρήστες μπορούν να ορίσουν τον αριθμό των μηχανημάτων και τον ρόλο του καθενός. Έπειτα, όσον αφορά την σωλήνωση εισόδου, το μέγεθος μικρό-ομάδας στην διαδικασία batching πολλαπλασιάζεται με τον αριθμό των διαθέσιμων μηχανημάτων και γίνεται καθολικό ώστε να διαμοιραστεί, καθώς επίσης και τα δεδομένα εισόδου μοιράζονται σε κομμάτια (sharding) στα μηχανήματα για να διαιρεθεί ο φόρτος εργασίας. Τέλος, συναρτήσεις όπως η map ή η prefetch ρυθμίζονται κατάλληλα ώστε να εκμεταλλεύονται τον διαθέσιμο παραλληλισμό με τα μηχανήματα του cluster και τα νήματα τους, προσφέροντας και τον επιπλέον μετασχηματισμό *interleave* ο οποίος παραλληλοποιεί το βήμα φόρτωσης δεδομένων, ενθέτοντας περιεχόμενα από διαφορετικές συλλογές, βελτιώνοντας έτσι την απόδοση στην περίπτωση πολλών διαφορετικών αρχείων προς είσοδο.

3.2 Σύστημα Spark

Το Apache Spark [43], [44] είναι μία **ενοποιημένη μηχανή για κατανεμημένη επεξεργασία δεδομένων**. Άλλα μοντέλα προγραμματισμού σε συμπλέγματα (clusters) που στοχεύουν σε ποικίλα υπολογιστικά φορτία είναι συνήθως αρκετά εξειδικευμένα, όπως το MapReduce [45] για επεξεργασία σε παρτίδες (batches), το Dremel της Google [46] η το Impala του Hadoop [11] για διαδραστικά ερωτήματα σε SQL, το Pregel [47] για υπολογισμούς σε γράφους και το Storm [48] για επεξεργασία δεδομένων σε ροές. Συνεπώς, αυτή η ενοποιημένη μηχανή παρέχει πλεονεκτήματα όπως ευκολότερη ανάπτυξη, αποτελεσματικό συνδυασμό διαφορετικών τύπων υπολογισμών που συνήθως γίνονται εντός-μνήμης RAM (in-memory) χωρίς την ανάγκη ενδιάμεσων αποθηκευτικών δομών για την επικοινωνία δεδομένων μεταξύ συστημάτων. Το Spark παρέχει μια Διεπαφή Προγραμματισμού Εφαρμογών (API) στις γλώσσες Scala, Java, Python, και R και μπορεί να χρησιμοποιήσει διάφορα εξωτερικά συστήματα για αποθήκευση δεδομένων, το πιο σύνηθες είναι το σύστημα αρχείων σε clusters HDFS (Hadoop Distributed File System) [49]. Είναι σχεδιασμένο να λειτουργεί ανεξάρτητα από το σύστημα αποθήκευσης που χρησιμοποιεί και έτσι οι χρήστες μπορούν να εκμεταλλευτούν υπάρχοντα δεδομένα και να συνδυάσουν διαφορετικές πηγές.



Εικόνα 3.6: Επιμέρους δομικά στοιχεία του συστήματος Spark [50]

Επιμέρους Βιβλιοθήκες

Το ευρύτερο σύστημα του Spark εμπεριέχει κάποια υποσυστήματα για τα διαφορετικά είδη διαδικασιών και υπολογιστικών φορτίων. Αυτά είναι τα:

1. **Spark Core**
   Το Spark Core [51] είναι η βάση του συστήματος. Εδώ είναι ενσωματωμένη η λειτουργικότητα των RDDs και παρέχονται APIs στις γλώσσες Scala, Java, Python και R. Επιπλέον, προσφέρει λειτουργικότητες για εντός-μνήμης υπολογισμούς όπως διαχείριση μνήμης, χρονοδιάγραμμα εργασιών, ανακατάταξη δεδομένων και ανάκτηση μετά από σφάλμα.

2. **Spark SQL**
   Το Spark SQL [52] ενσωματώνει διαδικασίες σχεσιακής επεξεργασίας σε ένα πιο γενικού σκοπού ενοποιημένο σύστημα. Η διαδραστικότητα και η ευχέρεια που προσφέρει βασίζονται σε δυο πολύ βασικά δομικά στοιχεία, τα *DataFrames* και τον βελτιστοποιητή (optimizer) *Catalyst*.
   Τα DataFrames είναι συλλογές από δομημένες εγγραφές που μπορούν να χρησιμοποιηθούν είτε με την διαδικαστική είτε με την σχεσιακή διεπαφή του Spark. Ουσιαστικά, είναι ισοδύναμα με τους πίνακες μιας σχεσιακής βάσης δεδομένων. Μπορούν να δημιουργηθούν είτε από πίνακες μιας εξωτερικής πηγής δεδομένων είτε από υπάρχοντα RDDs. Τέλος, τα DataFrames είναι «τεμπέλικα» όπως τα RDDs καθώς κάθε τέτοιο αντικείμενο αντιπροσωπεύει ένα λογικό πλάνο υπολογισμού που δεν εκτελείται μέχρι να κληθεί διαδικασία «εξόδου» όπως save, παρέχοντας έτσι και δυνατότητα ανάκτησης χαμένων δεδομένων.
   Ο βελτιστοποιητής Catalyst είναι βασισμένος σε δομές συναρτησιακού προγραμματισμού της γλώσσας Scala. Σε γενικές γραμμές, λειτουργεί με δενδρικές δομές εφαρμόζοντας κανόνες για τον χειρισμό τους. Παρέχει βιβλιοθήκες για επεξεργασία σχεσιακών ερωτημάτων και χειρίζεται φάσεις εκτέλεσης όπως ανάλυση, λογική βελτιστοποίηση, φυσικό πλάνο και παραγωγή κώδικα.

3. **Spark Streaming**
   Το Spark Streaming [53] στοχεύει στην επεξεργασία δεδομένων πραγματικού χρόνου σε ροές. Τα εισερχόμενα δεδομένα έρχονται είτε από πηγές ροών είτε από στατικές δομές όπως βάσεις δεδομένων χρησιμοποιώντας μια διακριτοποιημένη ροή (DStream) που απεικονίζεται από RDDs. Τα δεδομένα μαζεύονται με την μορφή μικρο-ομάδων (micro-batches) προκειμένου να γίνει η επεξεργασία όσο πιο κοντά σε πραγματικό χρόνο γίνεται.

4. **Spark MLlib / ML**
   Το Spark MLlib [54] είναι μια κατανεμημένη βιβλιοθήκη για μηχανική μάθηση που επωφελείται από τον παραλληλισμό των δεδομένων και των μοντέλων. Βασίζεται στο δομικό στοιχείο RDD αλλά σε νεότερες εκδόσεις αντικαθίσταται σταδιακά από την βιβλιοθήκη ML που βασίζεται στα DataFrames του Spark SQL.
   Η βιβλιοθήκη παρέχει πολλές κατανεμημένες υλοποιήσεις αλγορίθμων μάθησης όπως γραμμικά μοντέλα, απλοϊκό Bayes, δένδρα απόφασης, ομαδοποίηση k-μέσων και πολλές άλλες. Επίσης, προσφέρει την δυνατότητα σωλήνωσης (pipeline) σειράς διαδικασιών όπως προ-επεξεργασία δεδομένων, εξαγωγή χαρακτηριστικών, εκπαίδευσης και επαλήθευσης, επωφελούμενη ταυτόχρονα από την κατανεμημένη και αποδοτική διαχείριση μεγάλου όγκου δεδομένων που προσφέρει το Spark.

5. **Spark GraphX**

Το Spark GraphX [55] είναι ένα σύστημα υπολογισμού γράφων και βασίζεται στο βασικό δομικό στοιχείο του Spark, τα RDDs, επεκτείνοντας τα σε Resilient Distributed Graphs (RDG) όπου οι εγγραφές συσχετίζονται με κόμβους και ακμές σε έναν γράφο. Χαρακτηριστικά οφέλη σε αυτήν την ενσωμάτωση περιλαμβάνουν ανοχή σε σφάλματα, παραλληλισμό δεδομένων και στοιχείων των γράφων και αναπαράσταση σε πίνακες της τομής-κόμβων των γράφων.

Resilient Distributed Dataset – RDD

Το βασικό δομικό στοιχείο της διεπαφής προγραμματισμού Spark λέγεται Resilient Distributed Dataset ή RDD [56]. Είναι μια συλλογή δεδομένων μόνο για ανάγνωση (read-only) διαμοιρασμένη σε πολλούς κόμβους. Ένα RDD μπορεί να δημιουργηθεί είτε από αποθηκευμένα δεδομένα είτε από άλλα προ-υπάρχοντα RDD. Τα RDDs δεν έχουν πάντα «υλική» υπόσταση. Αντιθέτως, κρατούν απαραίτητες πληροφορίες της κατασκευής τους από σταθερά δεδομένα και τους μετασχηματισμούς που έγιναν στην πορεία. Αυτές οι πληροφορίες ονομάζονται καταγωγή (lineage) και παρέχουν ανοχή σε σφάλματα και αποδοτική ανάκτηση δεδομένων σε περίπτωση απώλειας κάποιου διαμερισμού (partition) του RDD. Τα RDDs μπορούν να διατηρηθούν για επαναχρησιμοποίηση είτε στην RAM είτε στον δίσκο, σε περίπτωση που ο όγκος δεδομένων είναι πολύ μεγάλος ή ο χρήστης το διευκρινίσει.

Το Spark υποστηρίζει δύο τύπους λειτουργιών πάνω σε RDD:

1. **Μετασχηματισμοί (Transformations)**

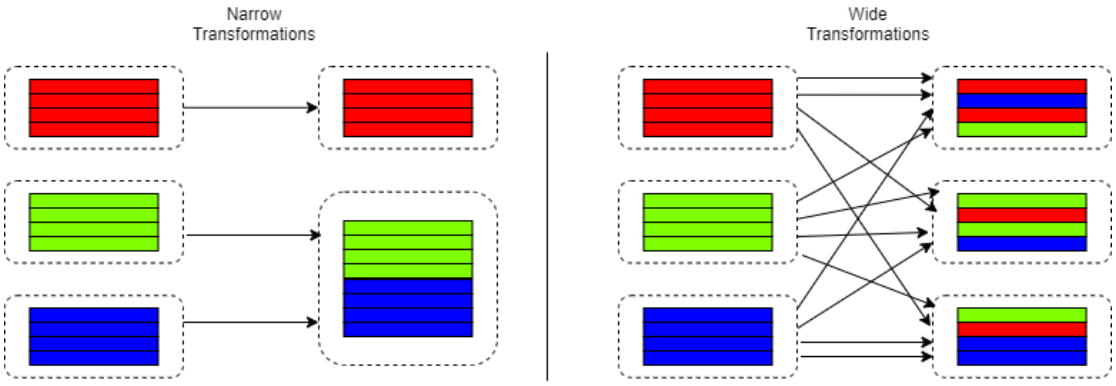
Οι μετασχηματισμοί είναι χονδρικές (coarse-grained) λειτουργίες είτε σε σταθερά δεδομένα είτε σε προ-υπάρχοντα RDDs και παράγουν σαν αποτέλεσμα καινούρια RDDs. Ονομάζονται «τεμπέλικες» (lazy) λειτουργίες καθώς δεν πυροδοτούν κάποιον υπολογισμό μέχρι να συμβεί κάποια πράξη (action). Με αυτόν τον τρόπο μπορούν να σωληνωθούν (pipeline) και να βελτιστοποιηθούν εσωτερικά, αυξάνοντας σημαντικά την αποδοτικότητα. Οι μετασχηματισμοί κατηγοριοποιούνται σε δύο είδη με βάση τις αλληλεξαρτήσεις μεταξύ των RDDs:

a) <u>Στενοί μετασχηματισμοί (Narrow Transformations)</u>

Κάθε διαμερισμός του «πατέρα» RDD χρησιμοποιείται το πολύ από έναν διαμερισμό του «παιδιού» RDD.
Παραδείγματα στενών μετασχηματισμών είναι οι συναρτήσεις map, filter και union.

b) <u>Ευρείς μετασχηματισμοί (Wide Transformations)</u>

Πολλαπλοί διαμερισμοί των «παιδιών» RDD μπορεί να εξαρτώνται από κάθε διαμερισμό του «πατέρα» RDD.
Παραδείγματα ευρέων μετασχηματισμών είναι οι συναρτήσεις groupByKey και join.
Στην Εικόνα 3.7 φαίνονται σχηματικά οι στενοί (α) και ευρείς (β) μετασχηματισμοί.



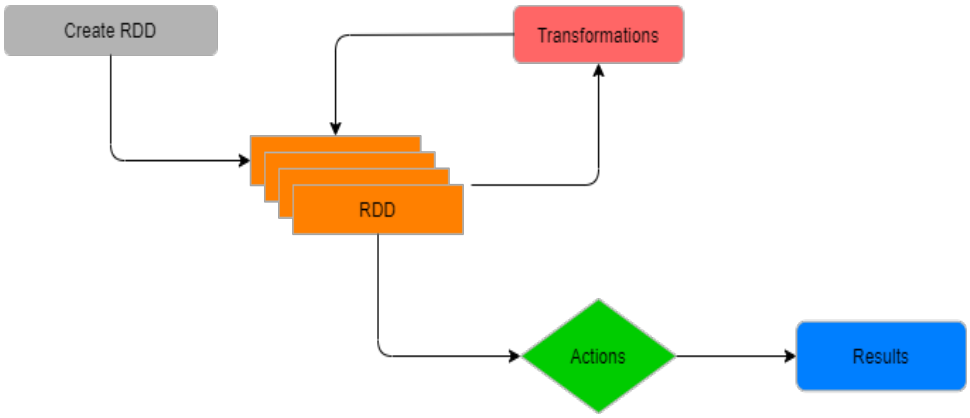
(α) Στενοί Μετασχηματισμοί (β) Ευρείς Μετασχηματισμοί

Εικόνα 3.7: Μετασχηματισμοί σε RDD του Spark

2. Πράξεις (Actions)

Οι πράξεις πυροδοτούν υπολογισμούς για να επιστρέψουν τιμές στο πρόγραμμα ή να αποθηκεύσουν δεδομένα σε εξωτερικό σύστημα. Δεν δημιουργούν καινούρια RDDs όπως οι μετασχηματισμοί. Σε αυτό το σημείο, το Spark σταματάει την «τεμπέλικη» εκτέλεση και χρησιμοποιεί τον γράφο καταγωγής (lineage graph) για να παράγει αποτέλεσμα. Παραδείγματα πράξεων είναι οι συναρτήσεις count, collect και save (που σαν αποτέλεσμα αποθηκεύει δεδομένα σε σύστημα αποθήκευσης).

Στην Εικόνα 3.8 παραθέτουμε τον κύκλο ζωής των RDDs από την δημιουργία τους προς την εξαγωγή αποτελεσμάτων.



Εικόνα 3.8: Κύκλος ζωής RDD

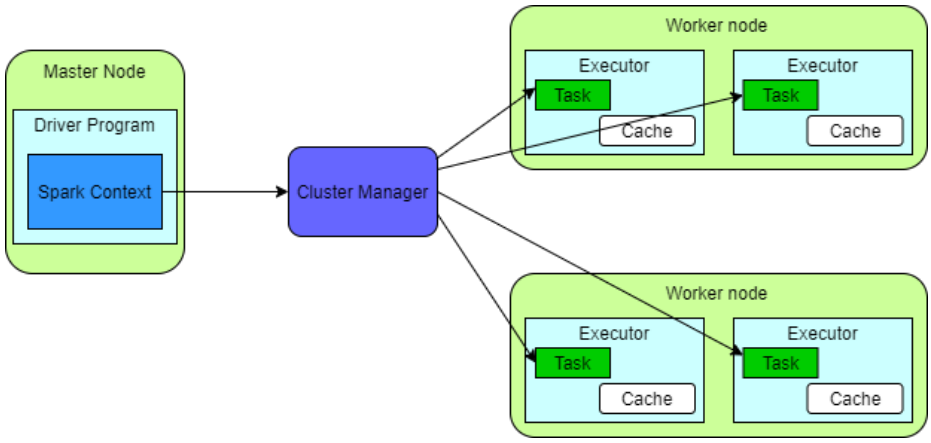
Αρχιτεκτονική – Λειτουργικότητα

Το σύστημα Spark [56] λειτουργεί κατά βάση σε σύμπλεγμα (cluster) μηχανών-διαδικασιών και απαρτίζεται από τα βασικά δομικά στοιχεία *πρόγραμμα-οδηγός (driver/master) και πρόγραμμα-εκτελεστής/εργάτης (executor/worker)*. Το σύστημα τρέχει πάνω από έναν *διαχειριστή συμπλέγματος (cluster manager)* δουλειά του οποίου είναι επίσης και ο διαμοιρασμός πόρων.

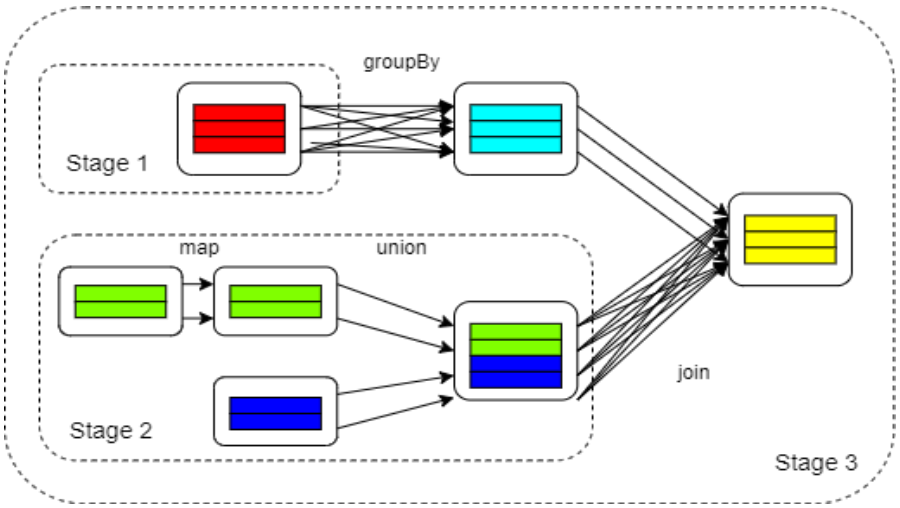
Με την εκκίνηση του συστήματος, σηματοδοτείται η δημιουργία του προγράμματος-οδηγού. Ένα σημαντικό δομικό στοιχείο που κουβαλάει και μεταφέρει δεδομένα και ρυθμίσεις ονομάζεται *SparkContext* και λειτουργεί σαν σημείο εισόδου για τις λειτουργικότητες του Spark. Μέσω του προγράμματος-οδηγού ορίζουν οι χρήστες τα επιθυμητά RDDs, τους

μετασχηματισμούς, τις πράξεις κλπ. Στην Εικόνα 3.9 μπορούμε να δούμε τα δομικά στοιχεία και την διασύνδεσή τους.

Έπειτα παίρνει σειρά ένα μέρος του συστήματος που ονομάζεται *υπεύθυνος προγράμματος (scheduler)* και καλείται να διαμερίσει τον φόρτο εργασίας σε *στάδια (stages)* σε μορφή *Κατευθυνόμενου Ακυκλικού Γράφου (Directed Acyclic Graph DAG)*. Σε αυτήν την εργασία γίνεται χρήση του γράφου καταγωγής (lineage graph) και γίνονται βελτιστοποιήσεις όπως σωλήνωση (pipeline) στενών μετασχηματισμών. Σύνηθες όριο ενός σταδίου είναι οι ευρείς μετασχηματισμοί. Τέλος, ο υπεύθυνος προγράμματος εκκινεί *εργασίες (tasks)* και τις αναθέτει στους εργάτες (workers) προκειμένου να υπολογίσουν τους απαραίτητους διαμερισμούς (partitions) των RDDs για να παραχθεί το τελικό αποτέλεσμα. Αυτές οι εργασίες ανατίθενται κατά κύριο λόγο με κριτήριο την τοπικότητα δεδομένων για καλύτερη απόδοση, αν δηλαδή ο επιθυμητός διαμοιρασμός δεδομένων υπάρχει ήδη εντός-μνήμης σε έναν κόμβο θα επιλεχθεί αυτός ο κόμβος για αυτήν την εργασία. Ένα παράδειγμα διαχωρισμού σε στάδια και επιμέρους ενεργειών του συστήματος φαίνεται στην Εικόνα 3.10.



Εικόνα 3.9: Δομικά στοιχεία και Αρχιτεκτονική του Spark



Εικόνα 3.10: Στάδια και επιμέρους μετασχηματισμοί και πράξεις

3.3 Σύστημα Horovod

Το Horovod [57] είναι ένα αυτόνομο πακέτο της Python που διευκολύνει την υλοποίηση αλγορίθμων κατανεμημένης βαθιάς μάθησης στα συστήματα Tensorflow, Keras [58], PyTorch [6] και Apache MXNet [5]. Ο βασικός στόχος του Horovod είναι να δέχεται έναν κώδικα εκπαίδευσης και να κατανέμει την εκτέλεσή του σε μία ή πολλές GPU ή και σε ομάδα μηχανημάτων με πολλαπλές GPU η καθεμία. Η επικοινωνία μεταξύ των μηχανημάτων επιτυγχάνεται με την *Διεπαφή Διαβίβασης Μηνυμάτων (Message Passing Interface MPI)* [59]. Η Διεπαφή Διαβίβασης Μηνυμάτων χρησιμοποιείται σε εφαρμογές που τρέχουν σε παράλληλες μηχανές με κατανεμημένη/διαμοιρασμένη μνήμη.

3.3.1 Διεπαφή Διαβίβασης Μηνυμάτων

Βασικό δομικό στοιχείο της διεπαφής είναι ο *δίαυλος επικοινωνίας (communicator)* ο οποίος ομαδοποιεί τις διαδικασίες δίνοντάς τους μοναδικές *βαθμίδες (ranks)* και διεκπεραιώνει την επικοινωνία μεταξύ τους. Μέσα σε κάθε ομάδα, ανατίθεται σε κάθε διαδικασία μία *τοπική βαθμίδα (local rank)* που υποδεικνύει την θέση της στον *εξυπηρετητή (host)*. Αν για παράδειγμα μία μηχανή λειτουργεί με 4 διαδικασίες πάνω σε 4 GPUs, σε κάθε GPU ανατίθεται μία μοναδική τοπική βαθμίδα μεταξύ 0 και 3. Ο συνολικός αριθμός των διαδικασιών σε μια ομάδα αναφέρεται ως *μέγεθος (size)*. Η επικοινωνία μεταξύ διαδικασιών επιτυγχάνεται με *ενέργειες αποστολής και λήψης (send / receive operations)*.

Το πρώτο είδος επικοινωνίας που γίνεται μεταξύ μόνο ενός αποστολέα και ενός παραλήπτη (διαδικασίες-ζευγάρια) ονομάζεται *σημείο-σε-σημείο (point-to-point)*. Αυτό το είδος επικοινωνίας επιτυγχάνεται με την χρήση (προαιρετικών) παραμέτρων *βαθμίδα-πηγή* και *ετικέτα μηνύματος* για να σαφηνιστεί ποιο μήνυμα προέρχεται από ποια διαδικασία.

Το δεύτερο είδος ονομάζεται *συλλογική επικοινωνία (collective communication)* και ενεργεί πάνω σε ομάδα διαδικασιών που δίνεται από τον δίαυλο επικοινωνίας. Το είδος αυτό χωρίζεται περαιτέρω σε δύο υποκατηγορίες: ενέργειες *μετακίνησης δεδομένων* και ενέργειες *καθολικού υπολογισμού*. Στην πρώτη ανήκουν 5 βασικοί τύποι ενεργειών: broadcast (μετάδοση δεδομένων από μια διαδικασία σε όλες τις άλλες), scatter (μετάδοση διακριτών δεδομένων από μια διαδικασία σε όλες τις άλλες, αλλιώς «ένας-προς-όλους προσωπική επικοινωνία»), gather (μία διαδικασία δέχεται είσοδο από όλες τις άλλες και τις συνδέει σειριακά) , all-gather (όπως η broadcast αλλά μετάδοση από κάθε διαδικασία προς όλες τις άλλες, καταλήγουν όλες με τα ίδια δεδομένα) και all-to-all (όπως η scatter αλλά μετάδοση από κάθε διαδικασία προς τις άλλες, αλλιώς «όλοι-προς-όλους προσωπική επικοινωνία»). Στην δεύτερη ανήκουν οι τύποι reduce (παραδείγματα: υπολογισμός μέγιστου ή αθροίσματος ενός σετ τιμών από διαφορετικές διαδικασίες) και scan (παράλληλη προσθήκη προθεμάτων σχετικά με μία διεργασία ορισμένη από τον χρήστη πάνω σε κατανεμημένα δεδομένα μιας ομάδας).

3.3.2 Horovod πάνω σε Spark

Το σύστημα Horovod προσφέρει την δυνατότητα να συνδυαστεί με το σύστημα Spark προκειμένου να κατανεμηθούν οι εργασίες εκπαίδευσης νευρωνικών δικτύων πάνω σε μηχανήματα συμπλέγματος Spark [60]. Οι φάσεις της προ-επεξεργασίας δεδομένων, της εκπαίδευσης και της αξιολόγησης του μοντέλου υλοποιούνται όλες μέσα στο Spark. Μέσω του Horovod προσφέρονται δύο Διεπαφές για τον χρήστη προκειμένου να εκτελεστούν οι διεργασίες του Horovod πάνω στα μηχανήματα-εργάτες του Spark, το υψηλού επιπέδου *Estimator API* και το πιο χαμηλού επιπέδου Run API. Το πρώτο ενδείκνυται, κυρίως, όταν η εκπαίδευση γίνεται σε δεδομένα πάνω σε δομές Dataframes του πακέτου Spark SQL, χρησιμοποιείται κάποια βιβλιοθήκη όπως η βιβλιοθήκη Keras του συστήματος TensorFlow και εφαρμόζεται κάποιο είδος αλγορίθμου καθόδου κλίσεων για την βελτιστοποίηση του μοντέλου. Αυτή η διεπαφή χρησιμοποιείται και στην συνέχεια στο πειραματικό

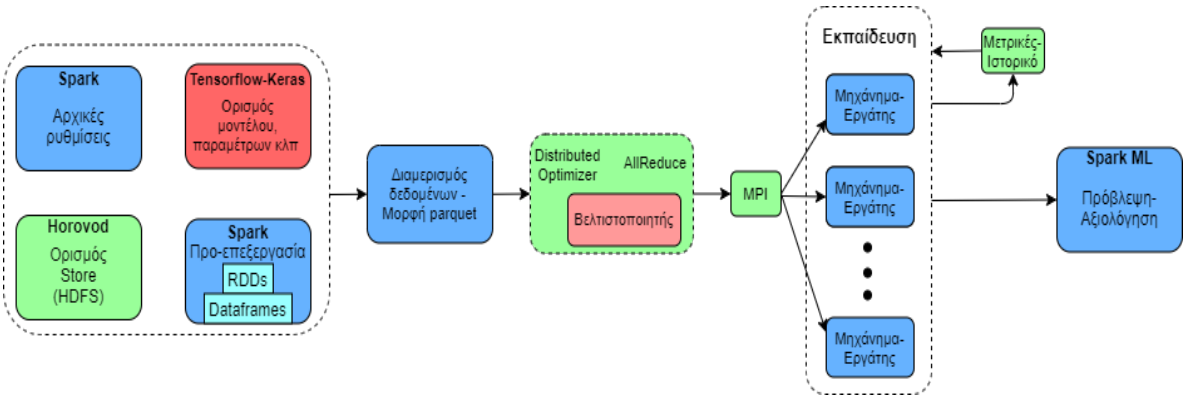
μέρος. Αυτό που κάνει είναι να εφαρμόζει τα Dataframes του Spark στον κώδικα εκπαίδευσης του νευρωνικού δικτύου και να αξιοποιεί το Horovod για να κατανείμει τον φόρτο στα μηχανήματα. Μετά από την εκπαίδευση, η διεπαφή επιστρέφει μια αναπαράσταση του μοντέλου η οποία μπορεί να χρησιμοποιηθεί μέσω της βιβλιοθήκης Spark MLlib για να γίνουν οι προβλέψεις και η αξιολόγηση του μοντέλου, πάλι πάνω στα Dataframes. Επιπλέον, παρέχει μία «έννοια» που ονομάζεται *Store* η οποία υποστηρίζεται για τοπικά συστήματα αρχείων αλλά και κατανεμημένα όπως το HDFS που αναφέρθηκε νωρίτερα. Χρησιμεύει για αποθήκευση ενδιάμεσων αναπαραστάσεων των δεδομένων εκπαίδευσης αλλά και για καταγραφή ιστορικού και καταγραφή μετρικών και επίδοσης μέσω εργαλείων όπως το Tensorboard του TensorFlow που αναλύεται σε επόμενη ενότητα.

Μια τυπική διαδικασία εκπαίδευσης με τον συνδυασμό των συστημάτων αυτών έχει ως εξής:

Γίνονται αρχικά οι απαραίτητες ρυθμίσεις και αρχικοποιήσεις για το Spark. Έπειτα ορίζεται ένα σημείο αποθήκευσης για την έννοια Store του Horovod, συνήθως το HDFS. Μετά ξεκινάει η φάση της προ-επεξεργασίας όπου το Spark διαβάζει τα δεδομένα και εφαρμόζει τους κατάλληλους μετασχηματισμούς στα RDDs ή τις αντίστοιχες πράξεις στα Dataframes προκειμένου να παραχθεί το τελικό σύνολο δεδομένων προς εκπαίδευση, το οποίο όμως πρέπει να είναι σε μορφή Dataframe. Για τον ορισμό του μοντέλου μπορεί να χρησιμοποιηθεί η βιβλιοθήκη Keras του συστήματος TensorFlow προκειμένου να οριστεί κατάλληλα το νευρωνικό δίκτυο, οι παράμετροί του και απαραίτητες συναρτήσεις όπως η συνάρτηση βελτιστοποίησης ή η συνάρτηση κόστους.

Στην φάση της εκπαίδευσης αξίζει να σημειωθούν μερικά σημαντικά σημεία στον συνδυασμό των δύο αυτών συστημάτων. Μέσω της διεπαφής Estimator το Horovod δίνει οδηγία στο Spark να διαμερίσει τα δεδομένα εκπαίδευσης και να τα μετατρέψει σε μορφή *parquet* [61], δηλαδή σε έναν τύπο δεδομένων με μορφή στηλών που είναι ιδανικός για εφαρμογές σε συστήματα του Hadoop. Σε επόμενο στάδιο το Horovod «τυλίγει» τον βελτιστοποιητή με μια μέθοδο DistributedOptimizer() προκειμένου να γίνει χρήση αλγορίθμου AllReduce για κατανεμημένο υπολογισμό κλίσεων. Τέλος το Horovod μέσω της διεπαφής MPI εκκινεί τις εργασίες εκπαίδευσης στα μηχανήματα του Spark cluster, μεταδίδοντας όλες τις αρχικές τιμές παραμέτρων σε όλες τις διεργασίες προκειμένου να επιτευχθεί συντονισμένη αρχικοποίηση και κρατώντας μετρικές για τους εργάτες στο τέλος κάθε εποχής που χρησιμεύουν για εργαλεία όπως το Tensorboard. Σημειώνεται ότι οι διεργασίες Horovod μπορούν να ανατεθούν είτε σε συσκευές CPU είτε GPU των μηχανημάτων του cluster, κάτι το οποίο συνδυάζεται με μία βασική ρύθμιση του Spark η οποία καθορίζει τον αριθμό των CPU που τους ανατίθεται κάθε εργασία. Για παράδειγμα αν έχουμε 2 μηχανήματα Spark με 4 πυρήνες CPU το καθένα, μπορούμε είτε να τρέξουμε 8 διεργασίες Horovod όπου κάθε διεργασία ανατίθεται σε έναν πυρήνα CPU είτε 4 διεργασίες πάνω σε 2 πυρήνες η καθεμία κλπ.

Τέλος, αφού ολοκληρωθεί η διαδικασία εκπαίδευσης χρησιμοποιείται η βιβλιοθήκη Spark ML για προβλέψεις και αξιολόγηση πάνω στα δεδομένα προς έλεγχο. Μπορούμε να δούμε σχηματικά την διαδικασία αυτή στην Εικόνα 3.11.



Εικόνα 3.11: Διαδικασία Εκπαίδευσης στο Spark μέσω Horovod

Κεφάλαιο 4

Μεθοδολογία

Στην παρούσα διπλωματική εργασία το βασικό μέρος είναι η σύγκριση σε ταχύτητα, απόδοση και βέλτιστη υλοποίηση της διαδικασίας κατανεμημένης εκπαίδευσης νευρωνικών δικτύων με χρήση δύο αρχιτεκτονικών που αναλύθηκαν νωρίτερα. Σκοπός αυτής της σύγκρισης είναι η απόφανση αν ο συνδυασμός των συστημάτων Spark και TensorFlow, που έχουν αμιγώς διαφορετική φιλοσοφία και αρχιτεκτονική, μπορεί να προσφέρει βελτίωση σε κάποιο κομμάτι όπως ταχύτητα, καλύτερη απόδοση σε μεγάλα clusters μηχανημάτων ή να συνδυάσει μαζί κάποια από τα προτερήματα του κάθε συστήματος. Στο κεφάλαιο αυτό θα γίνει μία εκτενής ανάλυση της μεθοδολογίας που ακολουθήθηκε. Στην ενότητα 4.1 περιγράφεται η υποδομή που στήθηκε, στην 4.2 αναλύονται τα σύνολα δεδομένων που χρησιμοποιήθηκαν στην εκπαίδευση και στην ενότητα 4.3 περιγράφεται η διαδικασία που ακολουθήθηκε στα δύο πειράματα και μερικά σημαντικά σημεία τους.

4.1 Υποδομή και Πειραματική Διάταξη

Η πειραματική αξιολόγηση πραγματοποιήθηκε απομακρυσμένα σε ένα cluster τριών μηχανημάτων του cloud «okeanos». Τα χαρακτηριστικά του κάθε μηχανήματος αναφέρονται στον Πίνακα 4.1.

Πίνακας 4.1: Χαρακτηριστικά μηχανημάτων του cluster

Χαρακτηριστικό	**Τιμή**
Μοντέλο Επεξεργαστή	Intel Core Processor (Skylake) @ 2.2 GHz
Πλήθος Πυρήνων	4
Νήματα / Πυρήνα	1
Μνήμη RAM	16GB

Χρησιμοποιήθηκαν τα συστήματα TensorFlow στην γλώσσα προγραμματισμού Python και Horovod πάνω σε Spark πάλι μέσω της γλώσσας Python, όπου για τον ορισμό και την εκπαίδευση των νευρωνικών δικτύων και στις δύο περιπτώσεις έγινε χρήση της βιβλιοθήκης Keras που παρέχεται από το TensorFlow. Στον Πίνακα 4.2 φαίνονται οι εκδόσεις των προαναφερθέντων συστημάτων.

Πίνακας 4.2: Εκδόσεις Συστημάτων

Σύστημα	**Έκδοση**
Spark	2.4.0
TensorFlow	2.2.0
TensorBoard	2.2.2
TensorBoard-plugin-profile	2.3.0
Horovod	0.20.3

4.2 Σύνολα Δεδομένων

Τα σύνολα δεδομένων που χρησιμοποιήθηκαν κατά την διαδικασία της εκπαίδευσης νευρωνικών δικτύων στα πειράματα είναι το **mnist** και το **cifar-10**.

- **MNIST**

Το MNIST είναι ένα σύνολο ασπρόμαυρων εικόνων που απεικονίζουν χειρόγραφα αριθμητικά ψηφία 0-9, οι εικόνες δηλαδή εντάσσονται σε 10 διακριτές κλάσεις. Στον Πίνακα 4.3 παρατίθενται τα χαρακτηριστικά του συνόλου αυτού και στην Εικόνα 4.1 ακολουθούν παραδείγματα εικόνας της κάθε κλάσης.

Πίνακας 4.3: Στοιχεία του συνόλου Mnist

Χαρακτηριστικό	**Τιμή**
Μέγεθος Εικόνας	28 X 28 X 1
Μέγεθος Συνόλου Εκπαίδευσης	60000
Πλήθος Κλάσεων Ταξινόμησης	10
Μέγεθος Συνόλου Αξιολόγησης	10000

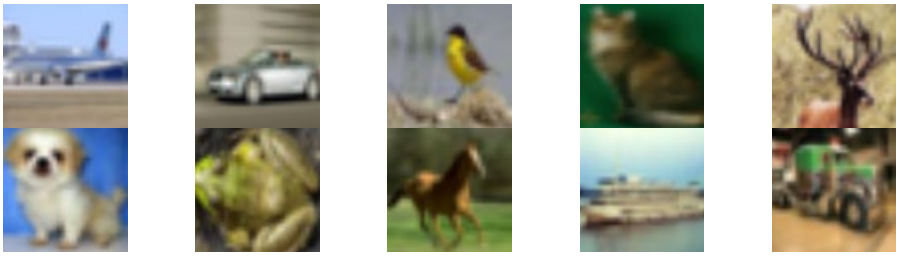
Εικόνα 4.1: Παραδείγματα εικόνων του συνόλου Mnist

- **CIFAR-10**

Το CIFAR-10 είναι ένα σύνολο έγχρωμων εικόνων που εντάσσονται σε διακριτές κλάσεις. Στον Πίνακα 4.4 παρατίθενται τα χαρακτηριστικά του συνόλου αυτού και στην Εικόνα 4.2 ακολουθούν παραδείγματα με εικόνες της κάθε κλάσης.

Πίνακας 4.4: Στοιχεία του συνόλου Cifar-10

Χαρακτηριστικό	**Τιμή**
Μέγεθος Εικόνας	32 X 32 X 3
Μέγεθος Συνόλου Εκπαίδευσης	50000
Εικόνες Εκπαίδευσης Ανά Κατηγορία	5000
Πλήθος Κλάσεων Ταξινόμησης	10
Μέγεθος Συνόλου Αξιολόγησης	10000
Εικόνες Αξιολόγησης Ανά Κατηγορία	1000



Εικόνα 4.2: Παραδείγματα εικόνων του συνόλου Cifar-10

4.3 Πειράματα

4.3.1 Φάσεις Πειραμάτων

Α) Πρώτη φάση

Στην πρώτη φάση πειράματος πραγματοποιήθηκε εκπαίδευση νευρωνικού δικτύου για διάφορες τιμές μεγέθους μικρό-ομάδας (batch size). Σε αυτήν την φάση κρατήθηκαν από τις εκτελέσεις οι επιμέρους χρόνοι που χρειάστηκε κάθε σύστημα για:

- να διαβάσει τα δεδομένα από το σημείο αποθήκευσής τους, συγκεκριμένα το HDFS
- να επεξεργαστεί και να φέρει τα δεδομένα σε κατάλληλη μορφή προς εκπαίδευση
- να ολοκληρώσει την διαδικασία εκπαίδευσης του νευρωνικού δικτύου

- να κάνει αξιολόγηση του δικτύου μέσω προβλέψεων στο σύνολο δεδομένων ελέγχου (test dataset)

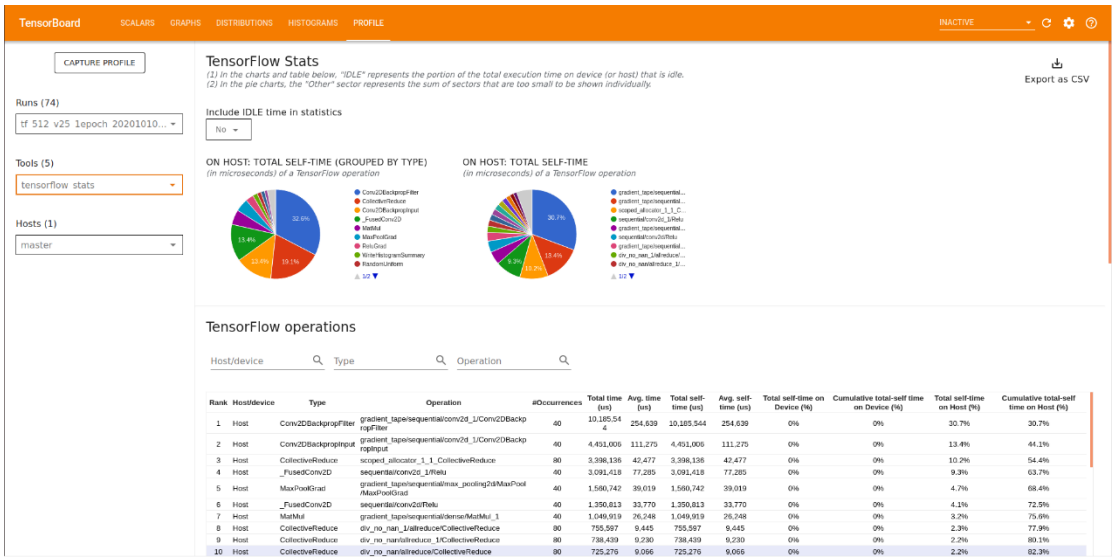
Στην διαδικασία αυτή δόθηκε προσοχή στα αποτελέσματα της συνάρτησης κόστους μετά από κάθε εποχή αλλά και στις τελικές τιμές ακρίβειας, τόσο στα δεδομένα εκπαίδευσης όσο και στα δεδομένα ελέγχου, στα δύο συστήματα προκειμένου να διασφαλιστεί ότι έχουν την ίδια συμπεριφορά όσον αφορά το ποιοτικό κομμάτι της εκπαίδευσης και να είναι ασφαλής και ουσιώδης η μετέπειτα σύγκρισή τους.

Παράλληλα, κατά την διάρκεια αυτών των εκτελέσεων, χρησιμοποιήθηκε ένα εξωτερικό εργαλείο, το Ganglia [62]. Το Ganglia συνδέεται με το κάθε μηχάνημα του cluster και καταγράφει σε ζωντανό χρόνο διάφορα μεγέθη σχετικά με το δίκτυο, τους επεξεργαστές, τα νήματα και άλλα σχετικά με την απόδοση του hardware του μηχανήματος. Με αυτό το εργαλείο μετρήθηκαν υπολογιστικά μεγέθη για το εκάστοτε όπως ποσοστά χρήσης των CPU και μέγεθος μνήμης που χρησιμοποιήθηκε μέσω διαγραμμάτων που κάλυπταν όλη τη διαδικασία.

Εικόνα 4.3: Διεπαφή Χρήστη με το εργαλείο Ganglia

Β) Δεύτερη φάση

Στην δεύτερη φάση πειράματος ακολούθησε μία πιο εις βάθος ανάλυση και διερεύνηση των συστημάτων προκειμένου να καταγραφούν αξιοσημείωτες διαφορές σε δομή, επιμέρους υπολογισμούς και βαθύτερη αρχιτεκτονική. Για αυτόν τον σκοπό έγινε χρήση του εργαλείου TensorBoard [63] μαζί με την επέκταση TensorFlow Profiler [64], [65] που προσφέρει το TensorFlow για σκιαγράφηση (profiling) και καταγραφή μετρικών και ιστορικού. Ορίζοντας τα διαστήματα που θα σκιαγραφήσει ο Profiler στην διαδικασία της εκπαίδευσης, καταγράφηκαν πληροφορίες σε διαστήματα μίας εποχής για αρκετές εποχές κατά την διάρκεια μίας εκτέλεσης. Με αυτόν τον τρόπο μελετήθηκαν περαιτέρω λεπτά σημεία που αφορούν για παράδειγμα τον χειρισμό δεδομένων, τους χρόνους εκτέλεσης συγκεκριμένων πράξεων-διαδικασιών και πολλά άλλα που θα αναλυθούν στο Κεφάλαιο 5.



Εικόνα 4.4: Διεπαφή Χρήστη με το εργαλείο TensorBoard

4.3.2 Στοιχεία και Παράμετροι Πειραμάτων

Στο **πρώτο πείραμα** έγινε χρήση ενός απλού συνελικτικού νευρωνικού δικτύου με τα εξής επιμέρους επίπεδα:

- συνελικτικό επίπεδο εισόδου με συνάρτηση ενεργοποίησης ReLU
- συνελικτικό επίπεδο με συνάρτηση ενεργοποίησης ReLU
- επίπεδο χωρικής υπό-δειγματοληψίας MaxPooling
- επίπεδο απόσυρσης με πιθανότητα 0.25
- επίπεδο αναδιαμόρφωσης σε μία διάσταση
- πλήρως συνδεδεμένο (πυκνό) επίπεδο 128 νευρώνων και συνάρτηση ενεργοποίησης ReLU
- επίπεδο απόσυρσης με πιθανότητα 0.5
- πλήρως συνδεδεμένο (πυκνό) επίπεδο εξόδου 10 νευρώνων με συνάρτηση ενεργοποίησης Softmax

Στο **δεύτερο πείραμα** χρησιμοποιήθηκε νευρωνικό δίκτυο αρχιτεκτονικής ResNet v2 με βάθος 56 (συνολικός αριθμός συνελικτικών επιπέδων). Στο δίκτυο αυτό τοποθετούνται στη σειρά τα εξής:

- συνελικτικό επίπεδο
- επίπεδο κανονικοποίησης με συνάρτηση ενεργοποίησης ReLU
- διακλάδωση μονοπατιών
    - στο ένα μονοπάτι υπάρχει μόνο ένα συνελικτικό επίπεδο

    - στο δεύτερο υπάρχει αρχικά ένα συνελικτικό και ακολουθούν δύο αλληλουχίες επιπέδων κανονικοποίηση-ενεργοποίηση-συνελικτικό που ονομάζονται επίπεδα συμφόρησης (bottleneck layers).

- τα δύο μονοπάτια αθροίζονται
- στην συνέχεια ακολουθούν 17 αλληλουχίες όπου στην κάθε αλληλουχία, το ένα μονοπάτι περνάει κατευθείαν προς άθροιση και το άλλο αποτελείται από 3 επίπεδα συμφόρησης.
- κανονικοποίηση με συνάρτηση ενεργοποίησης ReLU
- υπό-δειγματοληψία AveragePooling
- αναδιαμόρφωση σε μία διάσταση
- πλήρως συνδεδεμένο (πυκνό) επίπεδο εξόδου 10 νευρώνων με συνάρτηση ενεργοποίησης Softmax.

Στον Πίνακας 4.5, τέλος, αναγράφονται κάποιες τιμές παραμέτρων που αφορούν συνολικά την διαδικασία εκπαίδευσης.

Πίνακας 4.5: Παράμετροι πειραμάτων

Παράμετρος	**Πείραμα 1**	**Πείραμα 2**
Μεγέθη μικρό-ομάδων(cluster)	96 μέχρι 1536 (βήμα x2)	64 μέχρι 512 (βήμα x2)
Εποχές	30	25
Συνάρτηση κόστους	Categorical Cross-Entropy	Categorical Cross-Entropy
Βελτιστοποιητής	Adadelta	Adadelta

Κεφάλαιο 5

Αποτελέσματα και Σχολιασμός

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των πειραμάτων όπως αυτά αναλύθηκαν στο Κεφάλαιο 4. Στις ενότητες 5.1 και 5.2 παρουσιάζονται τα αποτελέσματα πάνω στα σύνολα δεδομένων εκπαίδευσης mnist και cifar-10 αντίστοιχα μαζί με σχόλια που προκύπτουν από αυτά τα δεδομένα.

5.1 Αποτελέσματα Mnist

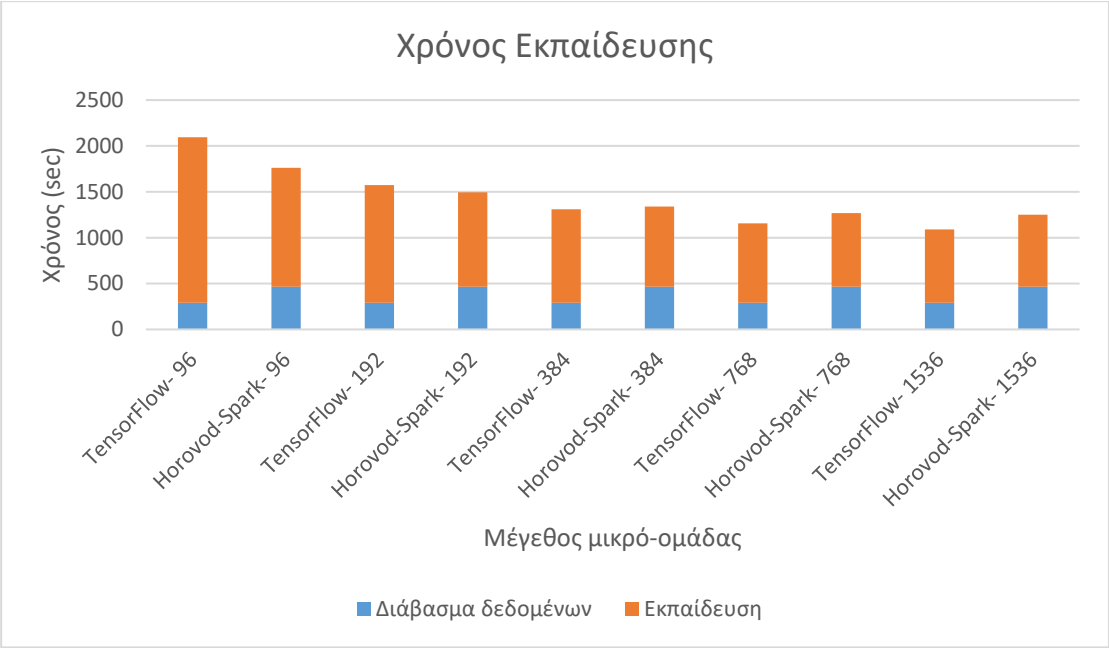
5.1.1 Πρώτη Φάση Πειράματος

Στο κομμάτι της εκπαίδευσης, αρχικά καταγράφηκε ο χρόνος που χρειάστηκε κάθε σύστημα προκειμένου να διαβάσει τα δεδομένα από το σημείο αποθήκευσης και να τα φέρει σε κατάλληλη μορφή προς εκπαίδευση. Σε αυτήν την φάση αξίζει να σημειωθούν τα παρακάτω:

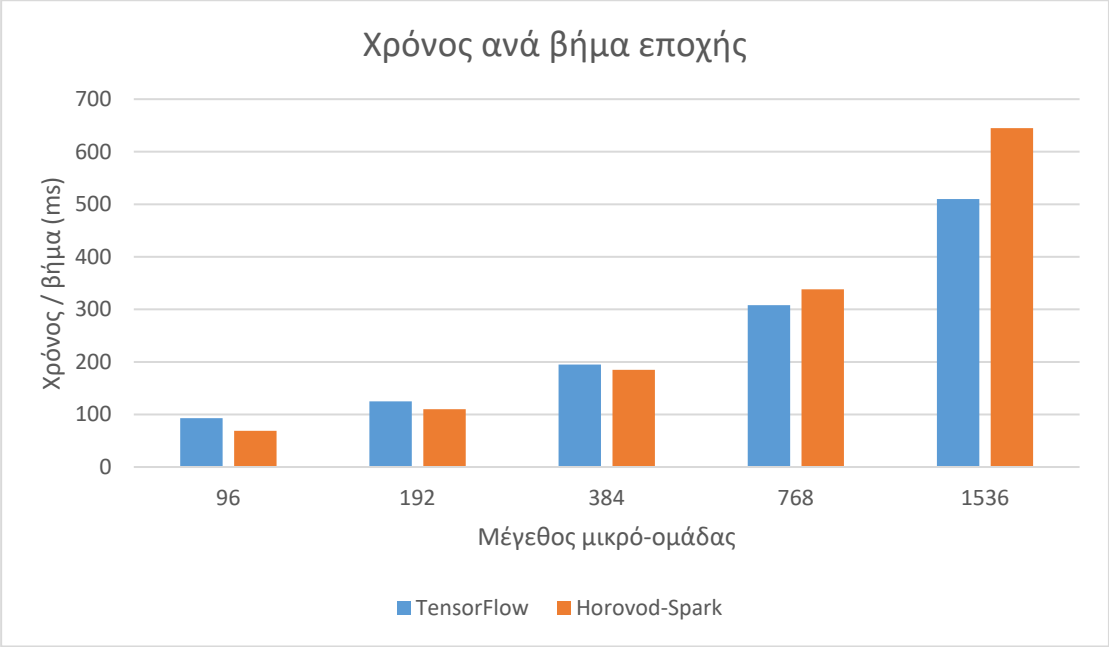
- στο **TensorFlow** ο χρόνος αυτός ενσωματώνεται στην διάρκεια της πρώτης εποχής. Εκεί μέσω του μετασχηματισμού cache στο pipeline, τα δεδομένα εκπαίδευσης διατηρούνται και είναι άμεσα διαθέσιμα στις επόμενες εποχές. Για αυτό το λόγο αυτός ο χρόνος μετρήθηκε σε ένα ξεχωριστό πρόγραμμα κάνοντας ένα πέρασμα των δεδομένων ομοίως με την εκπαίδευση μιας εποχής και καταγράφηκε **4 λεπτά και 51 δευτερόλεπτα**.

- στο **Spark** καταναλώνεται επιπλέον χρόνος καθώς χρειάζεται πριν ξεκινήσει η εκπαίδευση να μετασχηματιστούν τα δεδομένα σε μορφή Parquet όπως αναφέρθηκε στο Κεφάλαιο 3. Μετά από αυτόν τον μετασχηματισμό σε αντιστοιχία με τον μετασχηματισμό cache του TensorFlow, τα δεδομένα διατηρούνται στο Store για να είναι άμεσα διαθέσιμα. Συνολικά το Spark κατανάλωνε **7 λεπτά και 46 δευτερόλεπτα.**

Στην Εικόνα 5.1 φαίνεται ο χρόνος που χρειάστηκε κάθε σύστημα για την εκπαίδευση του νευρωνικού δικτύου για τις διάφορες τιμές μεγέθους μικρό-ομάδας. Στον χρόνο αυτό συμπεριλαμβάνεται και το διάβασμα και η προ-επεξεργασία των δεδομένων. Έπειτα στην Εικόνα 5.2 καταγράφεται κατά μέσο όρο ο χρόνος που καταναλώνει κάθε σύστημα για την εκπαίδευση μιας μικρό-ομάδας (mini batch), ο χρόνος ενός βήματος δηλαδή μέσα σε μια εποχή.



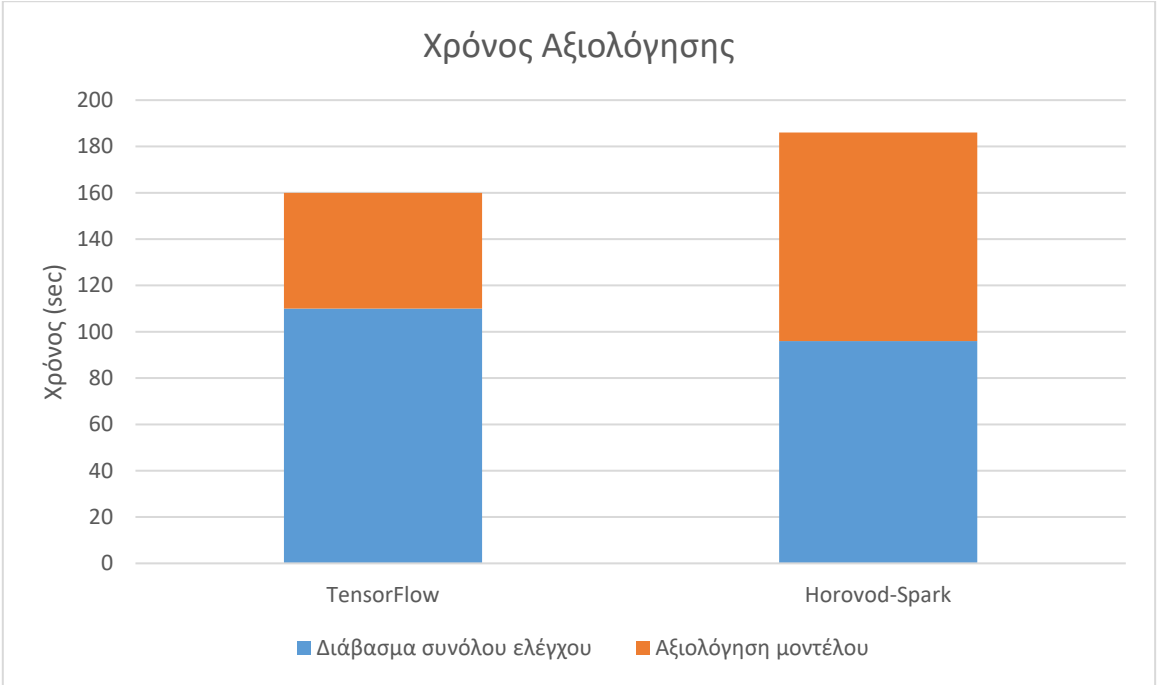
Εικόνα 5.1: Mnist - Διάγραμμα συνολικού χρόνου εκπαίδευσης



Εικόνα 5.2: Mnist - Διάγραμμα χρόνου εκτέλεσης βήματος εκπαίδευσης

Αντίστοιχα, όσον αφορά την φάση της αξιολόγησης, καταγράφηκε μετά την εκπαίδευση ο χρόνος που χρειάστηκε το κάθε σύστημα. Εδώ σημειώνονται τόσο οι χρόνοι διαβάσματος του συνόλου δεδομένων αξιολόγησης όσο και οι χρόνοι της αξιολόγησής τους μέσω του εκπαιδευμένου μοντέλου, οι οποίοι δεν επηρεάζονται από το μέγεθος μικρό-ομάδας..

Στην Εικόνα 5.1 φαίνονται αυτοί οι χρόνοι που καταγράφηκαν για τα δύο συστήματα:



Εικόνα 5.3 Mnist - Διάγραμμα χρόνου αξιολόγησης

Σημαντικό ρόλο στην ανάλυση και διερεύνηση της συμπεριφοράς των δύο συστημάτων παίζουν το μέγεθος μνήμης που καταναλώνεται καθώς και τα ποσοστά χρήσης της μονάδας CPU.

Αρχικά όσον αφορά την μνήμη, σημειώνεται ότι λόγω ρυθμίσεων και γενικής αρχιτεκτονικής του Spark μέσω του Hadoop, δεσμεύονται εξ αρχής κάποια κομμάτια μνήμης που χρειάζονται για τους κόμβους δεδομένων (datanodes) που διαχειρίζονται τα δεδομένα που προκύπτουν πάνω σε κάθε κόμβο και τους κόμβους ονομάτων (namenodes) που αναλαμβάνουν λειτουργίες σχετικές με αρχεία αλλά και την οργάνωση των datanodes. Για να είναι πιο ουσιαστική η παράθεση των μεγεθών, αφαιρέθηκαν αυτά τα κομμάτια από τις τιμές της χρήσης CPU και των δύο συστημάτων καθώς δεσμεύονται για όσο είναι ενεργό αυτό το σύστημα. Σημειώνεται επίσης ότι το Spark αναθέτει (ανάλογα την ρύθμιση που επιλέγει ο χρήστης) 1GB στο μηχάνημα driver και επίσης 1GB σε κάθε μηχάνημα-εργάτη (έτσι το ένα μηχάνημα που εκτελούσε και τους δύο ρόλους επωμίστηκε 2GB). Μπορούμε να δούμε στους παρακάτω πίνακες μια συνολική εικόνα, η οποία δεν είχε αισθητή διαφορά όσο αυξανόταν το μέγεθος μικρό-ομάδας καθώς δεν είναι πού μεγάλος ο όγκος δεδομένων προς εκπαίδευση ούτε τόσο πολύπλοκο το νευρωνικό δίκτυο.

Πίνακας 5.1: Mnist - Μνήμη συστήματος TensorFlow

	Cluster	**Master**	**Slave**
TensorFlow	7 GB	2.4 GB	2.4 GB

Πίνακας 5.2: Mnist - Μνήμη συστήματος Spark

	Cluster	**Master**	**Slave**
Spark	11.5 GB	4.5 GB	3.5 GB

Περνώντας, έπειτα, στην χρήση της CPU κατά την διάρκεια της εκπαίδευσης διακρίνονται τα εξής σημαντικά στοιχεία:

- για τις μικρές τιμές μεγέθους μικρό-ομάδας, το TensorFlow φαίνεται να μην αξιοποιεί καλά την CPU έχοντας χαμηλό ποσοστό χρήσης, όσο όμως αυξάνεται αυτό το μέγεθος αυξάνεται και η χρήση της CPU και αυτό φαίνεται κιόλας στο γεγονός ότι βελτιώνεται ο χρόνος εκπαίδευσης.

- στο Spark αντιθέτως η χρήση της CPU δεν ξεκινάει το ίδιο χαμηλά και έχει μια πιο ομαλή συμπεριφορά.

Στους παρακάτω πίνακες παρατίθενται αυτά τα δεδομένα γύρω από την χρήση της CPU στα δύο συστήματα κατά την διάρκεια εκπαίδευσης του νευρωνικού δικτύου.

Πίνακας 5.3: Mnist - Χρήση CPU κατά την εκπαίδευση στο TensorFlow

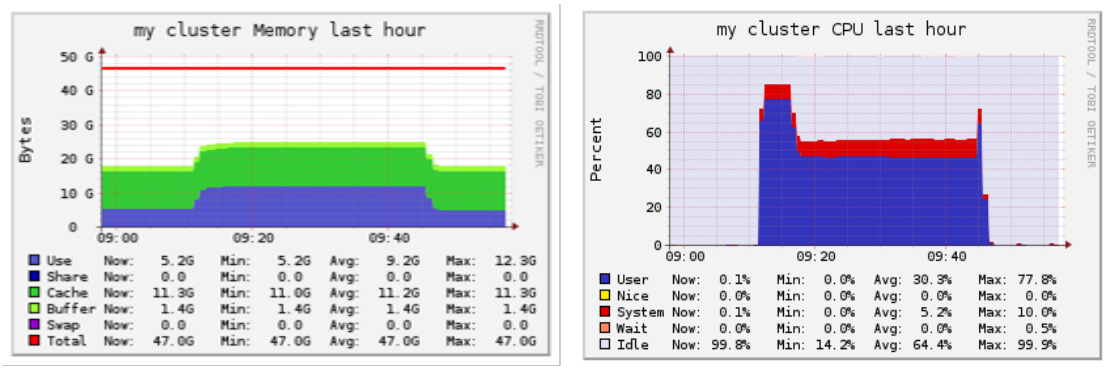
	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
TensorFlow	96	57.5%	57.5%	57.5%
	192	65%	65%	65%
	384	72.5%	72.5%	72.5%
	768	80%	80%	80%
	1536	85%	85%	85%

Πίνακας 5.4: Mnist - Χρήση CPU κατά την εκπαίδευση στο Spark

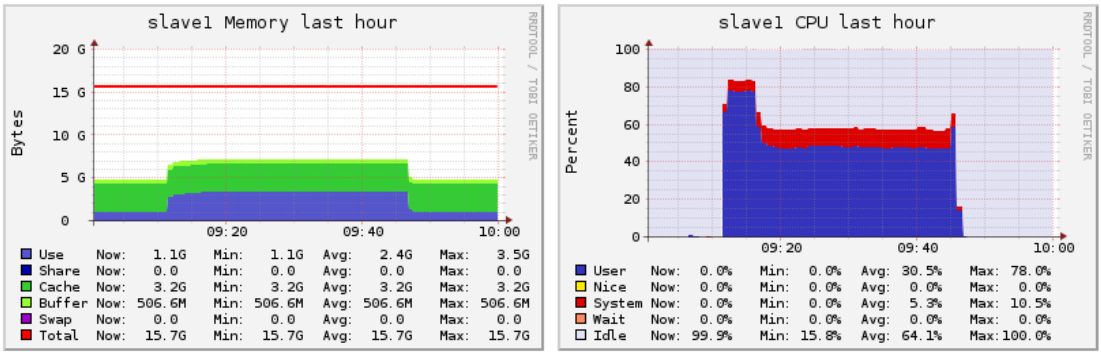
	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
Spark	96	70%	70%	70%
	192	75%	75%	72.5%
	384	77.5%	77.5%	77.5%
	768	82.5%	82.5%	82.5%
	1536	85%	85%	85%

Στην συνέχεια παρουσιάζονται μερικές εικόνες από το Ganglia σχετικά με την μνήμη και την CPU για τις διάφορες τιμές μικρό-ομάδας στα συστήματα TensorFlow και Spark.

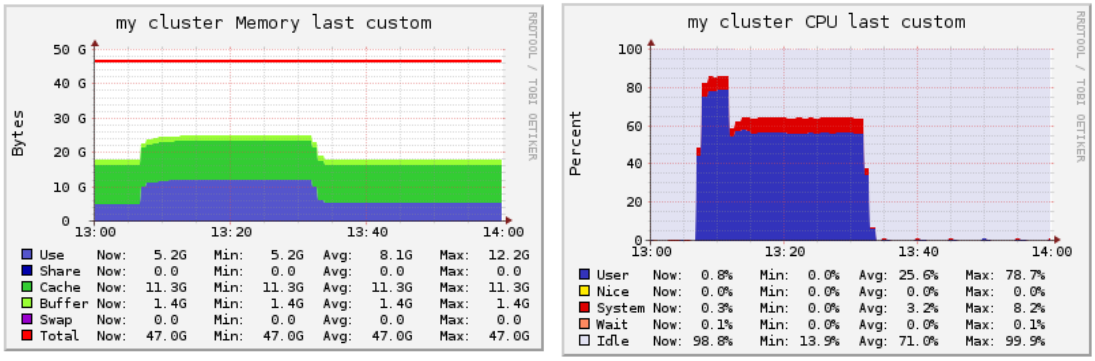
A) TensorFlow



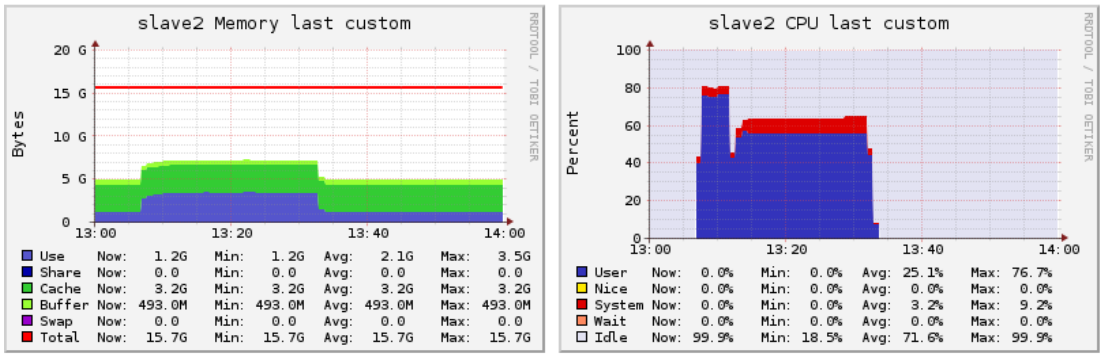
Εικόνα 5.4: Mnist - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 96**



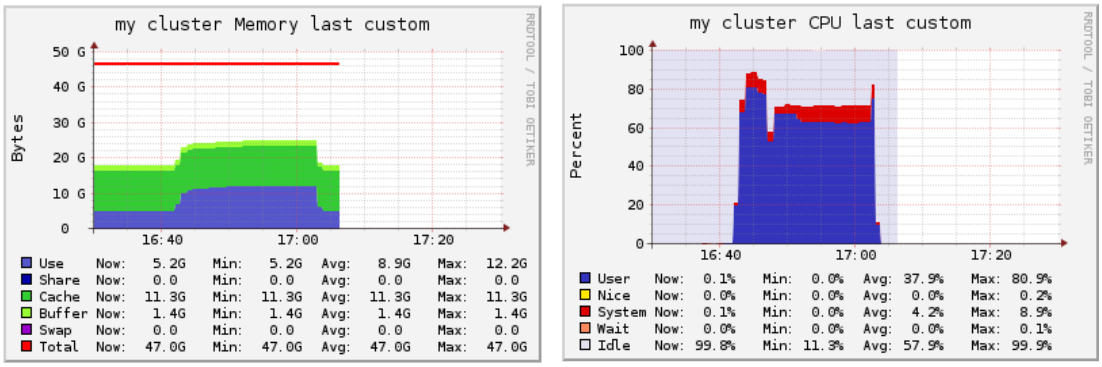
Εικόνα 5.5**:** Mnist - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 96**



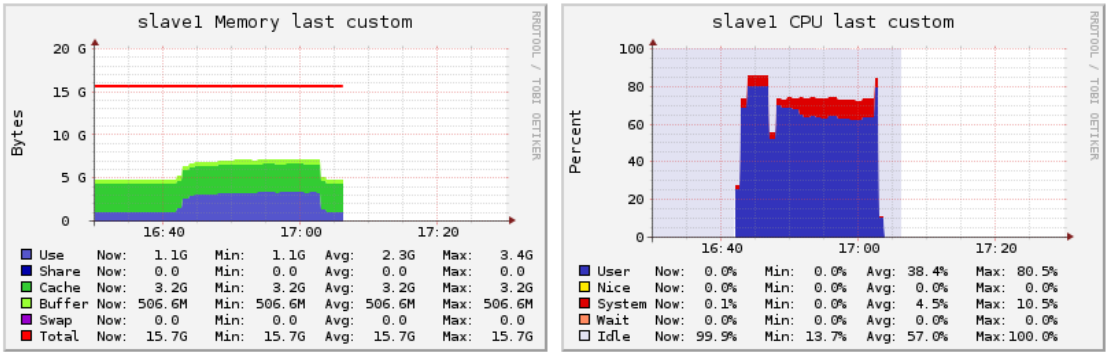
Εικόνα 5.6 Mnist - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 192**



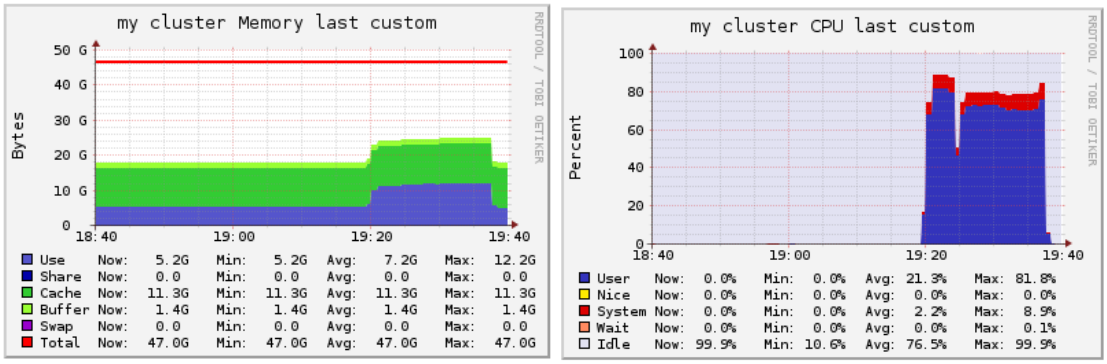
Εικόνα 5.7 Mnist - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 192**



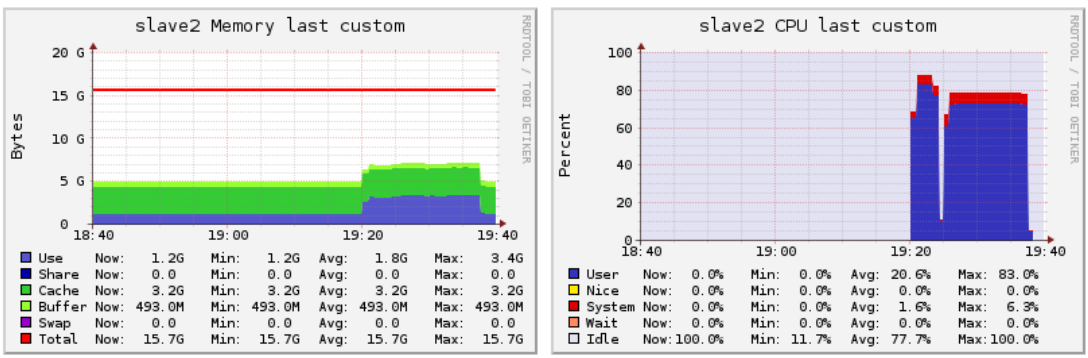
Εικόνα 5.8 Mnist - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 384**



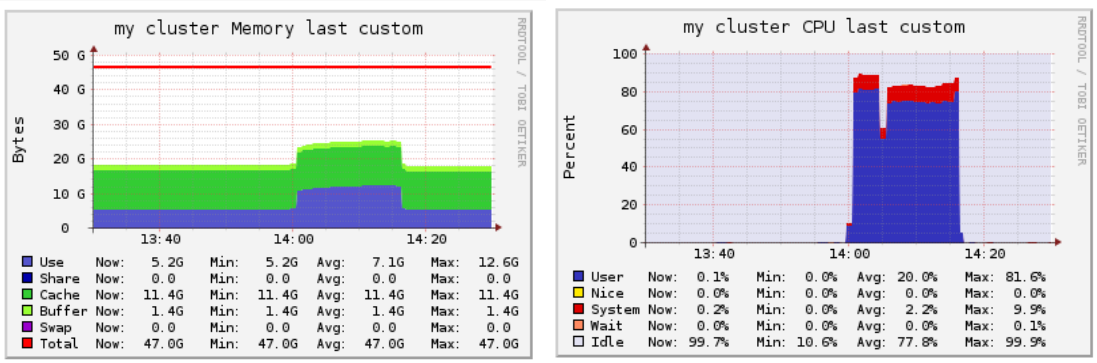
Εικόνα 5.9 Mnist - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 384**



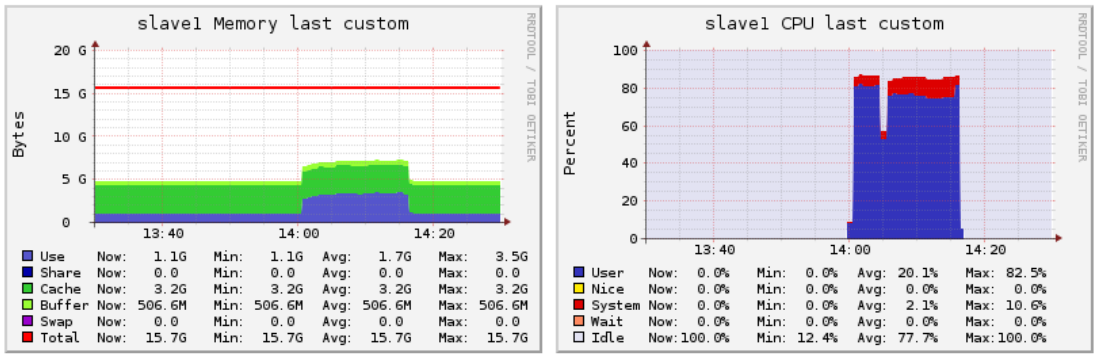
Εικόνα 5.10 Mnist - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 768**



Εικόνα 5.11 Mnist - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 768**

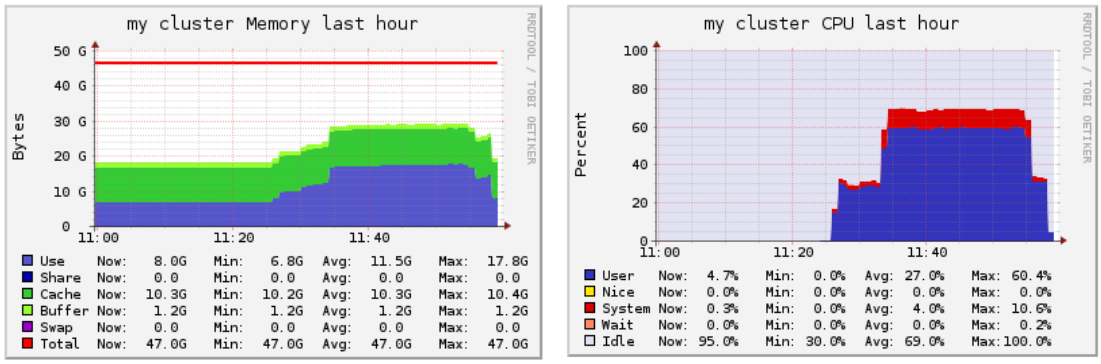


Εικόνα 5.12 Mnist - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 1536**

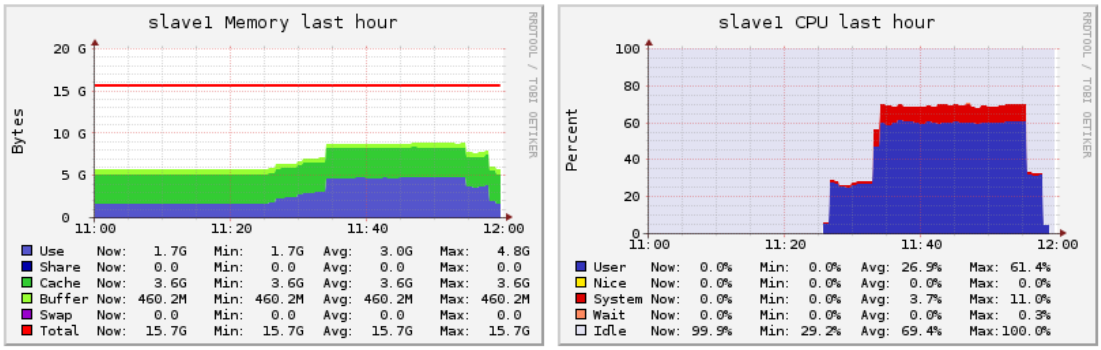


Εικόνα 5.13 Mnist - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 1536**

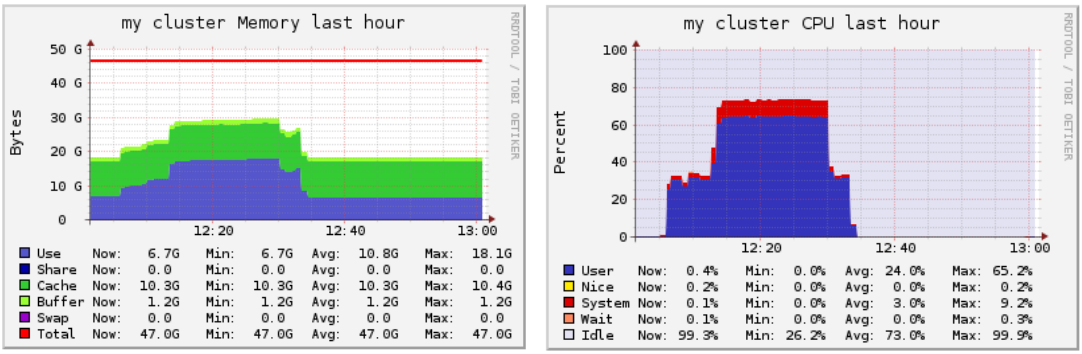
B) Spark



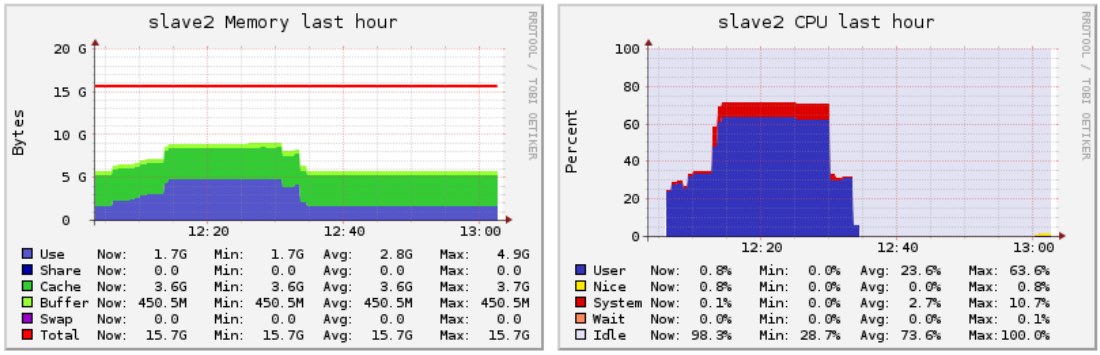
Εικόνα 5.14 Mnist - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 96**



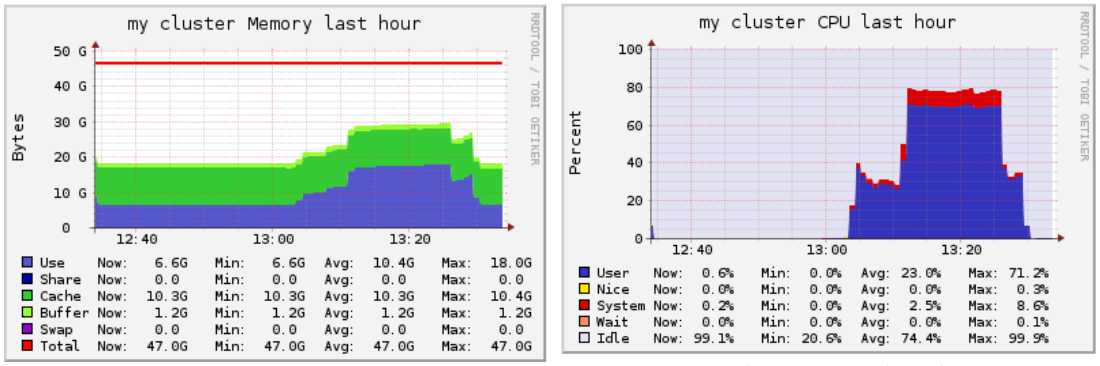
Εικόνα 5.15 Mnist - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 96**



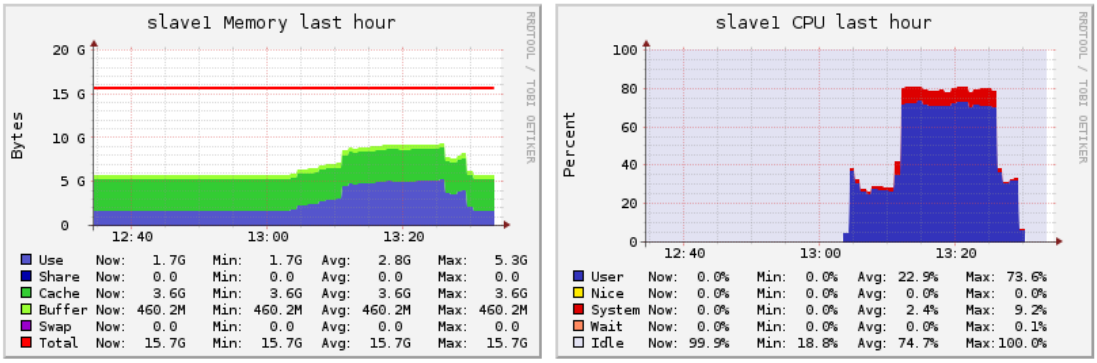
Εικόνα 5.16 Mnist - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 192**



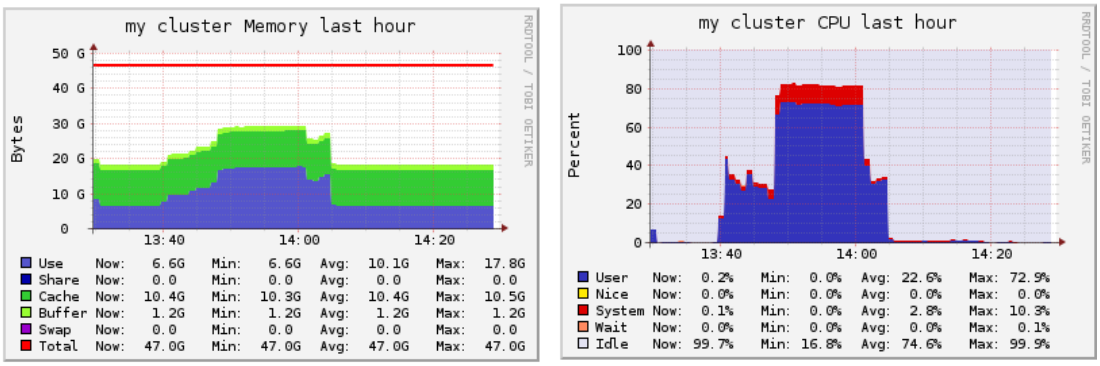
Εικόνα 5.17 Mnist - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 192**



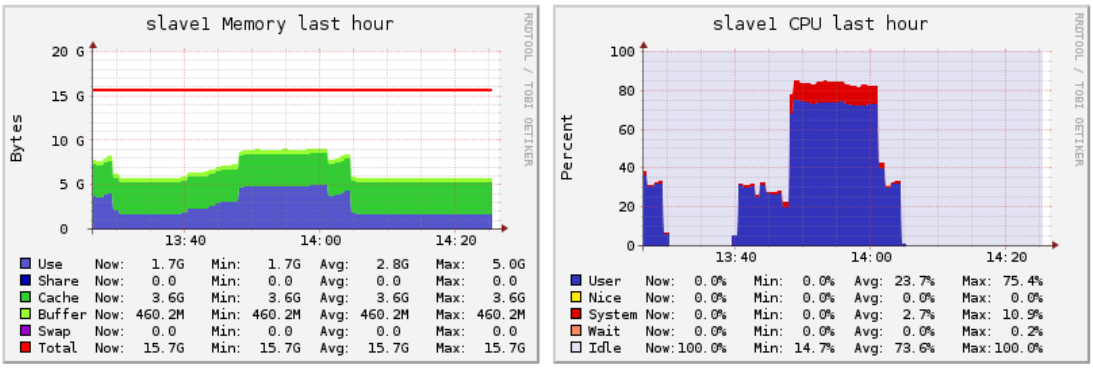
Εικόνα 5.18 Mnist - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 384**



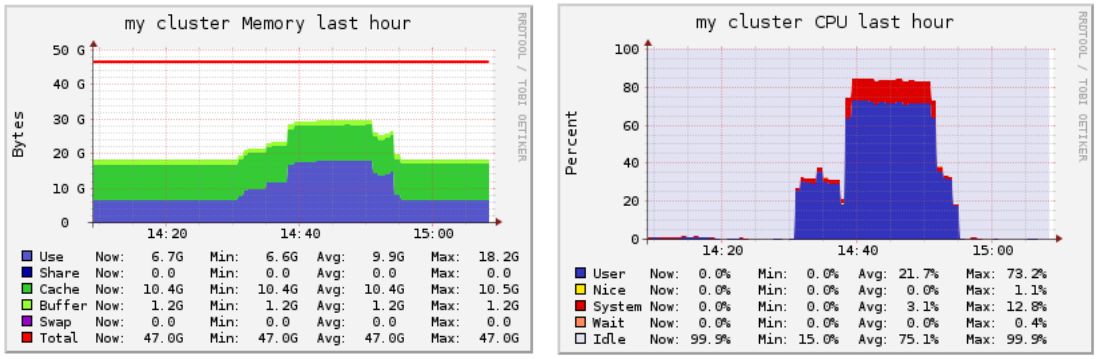
Εικόνα 5.19 Mnist - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 384**



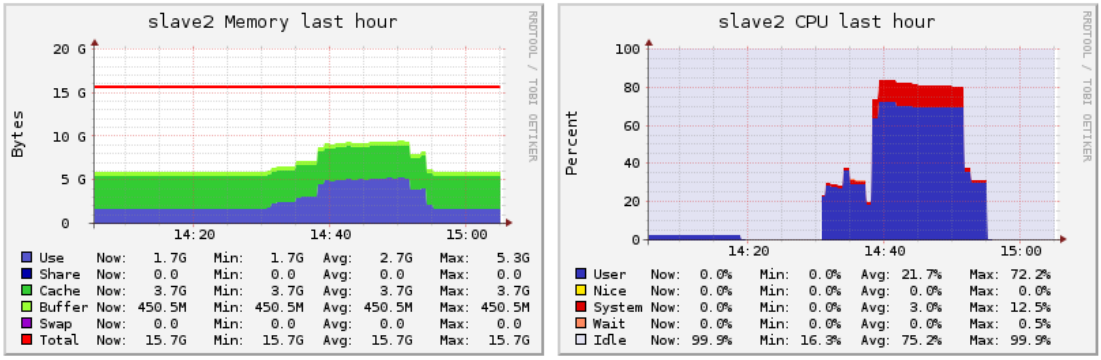
Εικόνα 5.20 Mnist - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 768**



Εικόνα 5.21 Mnist - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 768**



Εικόνα 5.22 Mnist - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 1536**



Εικόνα 5.23 Mnist - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 1536**

Μέσα από τα διαγράμματα αυτά, φαίνεται πιο ξεκάθαρα η συμπεριφορά του κάθε συστήματος στα διάφορα στάδια εκτέλεσης. Αρχικά, στα διαγράμματα μνήμης, διακρίνεται ότι από

την μία το TensorFlow ξεκινάει άμεσα την εκπαίδευση με το που γίνει διαθέσιμη η πρώτη μικρό-ομάδα, για αυτό και δεσμεύεται άμεσα η απαραίτητη μνήμη για το νευρωνικό δίκτυο, τις παραμέτρους και τα δεδομένα. Αυτό αντικατοπτρίζεται και στις υψηλές τιμές χρήσης CPU στην πρώτη εποχή εκπαίδευσης. Αντιθέτως στο Spark, η διαδικασία διαβάσματος και προ-επεξεργασίας είναι διακριτές από την εκπαίδευση και μάλιστα λόγω της «τεμπέλικης» φύσης του συστήματος, η απαραίτητη μνήμη δεσμεύεται σταδιακά όποτε κρίνεται απαραίτητο και μετά όταν αρχίζει η εκπαίδευση δεσμεύεται και η μνήμη για το δίκτυο. Ένα ακόμα σχόλιο που είναι απόρροια των προηγούμενων, είναι και η σύντομη απότομη πτώση στην χρήση CPU του TensorFlow για μεγάλα μεγέθη μικρό-ομάδας, η οποία οφείλεται στο γεγονός ότι η προ-επεξεργασία των επιμέρους κομματιών των δεδομένων από κάθε μηχάνημα μπορεί να διαφέρει λίγο χρονικά όσο αυξάνεται και το μέγεθος μικρό-ομάδας, κάτι το οποίο φαίνεται και για το Spark αλλά στο σημείο πριν αρχίσει η φάση της εκπαίδευσης.

5.1.2 Δεύτερη Φάση Πειράματος

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, στην δεύτερη φάση πειράματος έγινε χρήση του εργαλείου TensorBoard προκειμένου να γίνει σκιαγράφηση διάφορων βαθύτερων διαδικασιών και υπολογισμών. Η σκιαγράφηση αφορά βήματα/μικρό-ομάδες σε μία εποχή. Στην διπλωματική εργασία αυτή τα διαστήματα που μελετήθηκαν ήταν 5 ξεχωριστές εποχές κατά την διαδικασία εκπαίδευσης όπου κρατήθηκαν μέσοι όροι για τις διάφορες τιμές που προέκυψαν. Τα αποτελέσματα και τα σχόλια που ακολουθούν χωρίζονται σε κατηγορίες/ομάδες τελεστών (operators). Πρώτα παρατίθεται συνολικά το μέρος των δεδομένων με την μεγαλύτερη επιρροή και στην συνέχεια αναλύονται συγκεκριμένοι operators που εμφανίζουν αξιοσημείωτες διαφορές και σχόλια. Σημειώνεται ότι τα δεδομένα από την σκιαγράφηση μέσω του TensorBoard παρατίθενται σε μs ($10^{-3}\ sec$).

Πίνακας 5.5: Mnist - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 96

	96		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	21998564,60	21791115,60	1,01
Conv2DBackpropInput	9738674,00	9963119,20	0,98
_FusedConv2D	6455745,20	9177242,20	0,70
MatMul	6398953,00	6250161,60	1,02
MaxPoolGrad	1572841,00	1455096,80	1,08
ReluGrad	2305419,40	1838391,20	1,25
Dataset	1132499,60	533150,00	2,12
Mul	1234002,20	388340,00	3,18
RandomUniform	937047,20	945586,40	0,99
BiasAddGrad	1451272,60	1069852,80	1,36
_FusedMatMul	762696,60	773358,40	0,99
ResourceApplyAdadelta	2842334,00	2180331,60	1,30
MaxPool	471517,20	478579,20	0,99
Cast	399854,40	329226,80	1,21
GreaterEqual	163132,20	165556,00	0,99
PyFunc	346316,60	-	-
HorovodAllreduce/CollectiveReduce	189651,00	42556013,60	224,39

Πίνακας 5.6 Mnist - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 192

	192		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFLow**
Conv2DBackpropFilter	18168403,00	16686682,60	1,09
Conv2DBackpropInput	8685892,00	8574783,20	1,01
_FusedConv2D	5629445,60	6021299,60	0,93
MatMul	4647440,20	4705323,40	0,99
MaxPoolGrad	1534878,00	1459366,20	1,05
ReluGrad	1961334,20	1538163,60	1,28
Dataset	957173,20	363584,60	2,63
Mul	1118905,00	271002,20	4,13
RandomUniform	799144,80	808895,20	0,99
BiasAddGrad	1047995,00	733630,60	1,43
_FusedMatMul	759633,60	742298,00	1,02
ResourceApplyAdadelta	2354367,00	1099630,20	2,14
MaxPool	360411,20	357212,40	1,01
Cast	329050,00	258056,00	1,28
GreaterEqual	121358,40	118077,40	1,03
PyFunc	375892,00	-	-
HorovodAllreduce/CollectiveReduce	100855,40	22556609,80	223,65

Πίνακας 5.7 Mnist - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 384

	384		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	14556909,60	13131163,80	1,11
Conv2DBackpropInput	7070533,80	6237378,20	1,13
_FusedConv2D	5006691,60	6009654,60	0,83
MatMul	3351683,40	3388070,60	0,99
MaxPoolGrad	1311154,60	1537842,00	0,85
ReluGrad	1460132,40	1293941,60	1,13
Dataset	767513,60	269065,60	2,85
Mul	701004,00	291402,20	2,41
RandomUniform	707094,20	732003,00	0,97
BiasAddGrad	753256,40	590476,60	1,28
_FusedMatMul	537331,20	530398,00	1,01
ResourceApplyAdadelta	1151310,60	556821,60	2,07
MaxPool	314694,20	313218,20	1,00
Cast	218573,20	215927,40	1,01
GreaterEqual	89768,60	92078,00	0,97
PyFunc	340021,60	-	-
HorovodAllreduce/CollectiveReduce	52305,00	13611014,80	260,22

Πίνακας 5.8 Mnist - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 768

	768		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	13077715,20	11372150,80	1,15
Conv2DBackpropInput	5876030,40	4741629,60	1,24
_FusedConv2D	4840214,60	5213913,20	0,93
MatMul	2552739,60	2395021,60	1,07
MaxPoolGrad	1606640,80	1453984,80	1,10
ReluGrad	1239633,80	1058544,20	1,17
Dataset	733344,00	158907,80	4,61
Mul	524799,60	323996,60	1,62
RandomUniform	616402,00	624068,60	0,99
BiasAddGrad	665185,00	520061,00	1,28
_FusedMatMul	426525,40	373757,40	1,14
ResourceApplyAdadelta	654620,60	285860,00	2,29
MaxPool	272884,80	287266,00	0,95
Cast	249196,40	173234,60	1,44
GreaterEqual	82610,00	72566,40	1,14
PyFunc	310455,80	-	-
HorovodAllreduce/CollectiveReduce	30490,80	7088547,40	232,48

Πίνακας 5.9 Mnist - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 1536

	1536		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	13112160,00	10982612,80	1,19
Conv2DBackpropInput	5632470,80	4549663,00	1,24
_FusedConv2D	5049861,40	4434707,40	1,14
MatMul	2281958,80	1862222,00	1,23
MaxPoolGrad	2076198,40	1623936,60	1,28
ReluGrad	1183129,40	967803,00	1,22
Dataset	795339,00	81191,80	9,80
Mul	595540,00	369431,40	1,61
RandomUniform	573878,80	483132,80	1,19
BiasAddGrad	566950,00	474946,00	1,19
_FusedMatMul	397980,40	294456,60	1,35
ResourceApplyAdadelta	297190,40	147533,20	2,01
MaxPool	254620,80	227169,20	1,12
Cast	217894,40	140519,00	1,55
GreaterEqual	78477,20	58763,20	1,34
PyFunc	335731,80	-	-
HorovodAllreduce/CollectiveReduce	20068,60	5845230,20	291,26

Dataset

Στην κατηγορία αυτή ανήκουν τελεστές που αφορούν την διαχείριση των δεδομένων προς εκπαίδευση που τροφοδοτούν το νευρωνικό δίκτυο σε μικρό-ομάδες.

Αρχικά, όπως έχει αναφερθεί νωρίτερα, το TensorFlow καταναλώνει χρόνο στην πρώτη εποχή προκειμένου να γίνουν οι απαραίτητοι μετασχηματισμοί στα δεδομένα εκπαίδευσης (pre-processing) και στην συνέχεια αυτά τα δεδομένα είναι διαθέσιμα μέσω της cache. Σημειώνεται ότι το TensorFlow με το που έχει διαθέσιμα τα πρώτα μετασχηματισμένα δεδομένα εκκινεί την διαδικασία εκπαίδευσης και αυτά τα δύο γίνονται παράλληλα. Αντίστοιχα, το Spark κάνει όλους τους μετασχηματισμούς στην αρχή μαζί με τον μετασχηματισμό σε μορφή Parquet και το Horovod διατηρεί αυτά τα δεδομένα στο Store για την εκπαίδευση, με παρόμοια λογική όπως η cache. Στον Πίνακα 5.10 φαίνονται μαζεμένα τα δεδομένα που αφορούν αυτήν την κατηγορία.

Πίνακας 5.10: Mnist - Λόγος χρόνου Spark προς TensorFlow για τον operator Dataset

	96	**192**	**384**	**768**	**1536**
Spark / TensorFlow	2,12	2,63	2,85	4,61	9,80

Παρ' όλα αυτά, υπάρχει μια σημαντική διαφορά στα δύο συστήματα που επωφελεί αρκετά το TensorFlow. Το TensorFlow διαθέτει τον μηχανισμό Prefetch μέσω του οποίου η κάθε μικρό-ομάδα «ετοιμάζεται» για εκπαίδευση όσο τελειώνει η προηγούμενη, παράλληλα. Με αυτόν τον τρόπο δεν καταναλώνεται παρά ελάχιστος χρόνος όπου το σύστημα παραμένει ανενεργό περιμένοντας δεδομένα. Αντιθέτως το Spark πρώτα τελειώνει την επεξεργασία της προηγούμενης και μετά φροντίζει να φέρει την επόμενη μικρό-ομάδα. Επιπλέον, παρατηρήθηκε ότι στο Spark προστίθεντο επιπλέον φόρτος στην διαδικασία αυτή μέσω του operator PyFunc που αναφέρεται σε μια εσωτερική διαδικασία του Spark που αναλύεται παρακάτω.

Έχει αξία, επομένως, η σύγκριση των χρόνων **Dataset – Prefetch_Generator** (για TensorFlow) και **Dataset – PyFunc** (για Spark) με σκοπό να γίνει πιο καθαρή η συνεισφορά του μηχανισμού Prefetch.

Πίνακας 5.11: Mnist - Λόγος χρόνου Spark προς TensorFlow για τον operator Dataset ειδικεύοντας για Prefetch

	96	**192**	**384**	**768**	**1536**
Spark / TensorFlow	16,56	23,18	34,13	75,48	164,32

PyFunc

Σε αυτήν την κατηγορία που αφορά μόνο το Spark, εντάσσονται κάποιες εσωτερικές διαδικασίες του Spark οι οποίες εκτελούνται κατά την διάρκεια μιας εποχής τόσες φορές όσες και οι διαδικασίες Horovod που έχουν οριστεί, στην περίπτωσή μας 4. Συγκεκριμένα, η αιτία αυτού του επιπλέον φόρτου είναι η συνάρτηση to_petastorm_fn που καλείται από την get_or_create_dataset [60]. Οι μετασχηματισμοί, ειδικότερα, αφορούν την μετατροπή σε rdd, την διαδικασία map με την συνάρτηση to_petastorm_fn και τέλος την μετατροπή πίσω σε Dataframe και φαίνονται στην γραμμή :

```
df = df.rdd.map(to_petastorm).toDF()
```

Ποιοτικά, αυτό προκαλείται από την βιβλιοθήκη Petastorm [66]. Συνοπτικά, η βιβλιοθήκη Petastorm είναι ένα εργαλείο που χρησιμοποιεί εσωτερικά το σύστημα Horovod προκειμένου να μετατρέπει τα δεδομένα εκπαίδευσης στην κατάλληλη μορφή που απαιτεί το σύστημα που αναλαμβάνει την εκπαίδευση. Τα δέχεται σε μορφή Parquet και υποστηρίζει Tensorflow, Pytorch, και PySpark. Ενεργεί, δηλαδή σαν μεσολαβητής για την σύνδεση του κεντρικού συστήματος, εδώ το Spark, και της βιβλιοθήκης εκπαίδευσης, εδώ της Keras του TensorFlow. Το Petastorm, λοιπόν, δεν μπορεί να διαβάσει την δομή Vector του Spark και την μετατρέπει πρώτα σε πίνακα (Array). Το site αναφέρει : Convert Spark Vectors into arrays so Petastorm can read them.

Στην συγκεκριμένη περίπτωση, μετρήθηκε ότι προσθέτει κατά μέσο όρο **341 ms ανά εποχή** επιπλέον φόρτο.

AllReduce

Στην κατηγορία αυτή ανήκουν υπολογισμοί που αφορούν την συλλογή των gradients μέσω του αλγορίθμου AllReduce όπως αναλύθηκε σε προηγούμενο κεφάλαιο. Αρχικά έχουμε τα εξής δεδομένα για τα διάφορα μεγέθη μικρό-ομάδας:

Πίνακας 5.12: Mnist - Λόγος TensorFlow προς Spark για τον operator AllReduce

	96	**192**	**384**	**768**	**1536**
Spark / TensorFlow	0,0044	0,0044	0,0038	0,0043	0,0034

Κάνοντας, όμως, περισσότερη ανάλυση στους επιμέρους operators αυτής της κατηγορίας, προέκυψαν τα παρακάτω σχόλια και ενημερώσεις της σύγκρισης.

Όσον αφορά το TensorFlow, οι 4 operators :

- div_no_nan/allreduce/CollectiveReduce
- div_no_nan/allreduce_1/CollectiveReduce
- div_no_nan_1/allreduce/CollectiveReduce
- div_no_nan_1/allreduce_1/CollectiveReduce

παρατηρήθηκε ότι εκτελούνται σε ξεχωριστά νήματα, παράλληλα με τους άλλους operators, ενδιάμεσα στην εκπαίδευση της μικρό-ομάδας και δεν προσδίδουν επιπλέον χρονική επιβάρυνση. Όμως, ο operator **scoped_allocator_1_1_CollectiveReduce** εκτελείται στο τέλος, αφού τελειώσουν όλοι οι υπολογισμοί και πριν αρχίσει η ενημέρωση των παραμέτρων με τον βελτιστοποιητή (Adadelta), προσδίδοντας αισθητή χρονική επιβάρυνση καθώς το σύστημα μένει πρακτικά ανενεργό μέχρι να τελειώσει αυτός ο operator για να προχωρήσει στον βελτιστοποιητή. Συνοπτικά αναφέρεται ότι οι operators τύπου **scoped allocator** αποτελούν μία στατική βελτιστοποίηση που εφαρμόζει το TensorFlow όπου πολλαπλές διαδικασίες AllReduce συγχωνεύονται σε μία χρησιμοποιώντας προκαθορισμένους φορείς προσωρινής μνήμης (buffers) που χωράνε όλες τις εισόδους των διαδικασιών AllReduce.

Από την άλλη μεριά, το Horovod χρησιμοποιώντας την διεπαφή MPI εκτελεί **άμεσα** την διαδικασία AllReduce πάνω σε μεταβλητές που έχουν υπολογιστεί και προχωράει μάλιστα και στην εφαρμογή του βελτιστοποιητή, παράλληλα όπως και όλοι οι άλλοι operators. Αυτή η τεχνική ονομάζεται Tensor Fusion [57] και ουσιαστικά καθορίζει κάθε φορά μέσω φορέων προσωρινής μνήμης (buffers) μερικούς ταυνστές που μπορούν να συγχωνευτούν και να εκτελεστεί μία διαδικασία AllReduce αντί να γίνει μία διαδικασία για τον καθένα. Επιπλέον, παρατηρήθηκαν μερικοί operators τύπου AllReduce που ανήκαν στην κατηγορία Mul οπότε μπορούν να προστεθούν και αυτοί στην συνολική σύγκριση. (Σημειώνεται ότι αφαιρώντας αυτούς τους χρόνους από την κατηγορία Mul στο

Spark, προέκυψε ότι δεν αξίζει σχολιασμού στην περίπτωσή μας αυτό το κομμάτι, παρά τις μεγάλες τιμές του λόγου στους αρχικούς πίνακες)

Καταλήγουμε, επομένως, σε μια πιο καθαρή παράθεση operators AllReduce των δύο συστημάτων όπου στο TensorFlow υπολογίζουμε μόνο τον scoped_allocator και στο Horovod προσθέτουμε τους επιπλέον operators κάτω από την κατηγορία Mul. Στον Πίνακα 5.13 φαίνεται η τελική σύγκριση.

Πίνακας 5.13: Mnist - Ενημερωμένος λόγος TensorFlow προς Spark για τον operator AllReduce

	96	**192**	**384**	**768**	**1536**
Spark / TensorFlow	0,043	0,077	0,077	0,077	0,077

ResourceApplyAdadelta

Αναφέρεται συνοπτικά και η συγκεκριμένη κατηγορία operators καθώς αφορά τον βελτιστοποιητή Adadelta και φαίνεται ότι το Spark υστερεί όσον αφορά τον υπολογισμό και την ενημέρωση των βαρών.

Πίνακας 5.14: Mnist - Λόγος Spark προς TensorFlow για τον operator ResourceApplyAdadelta

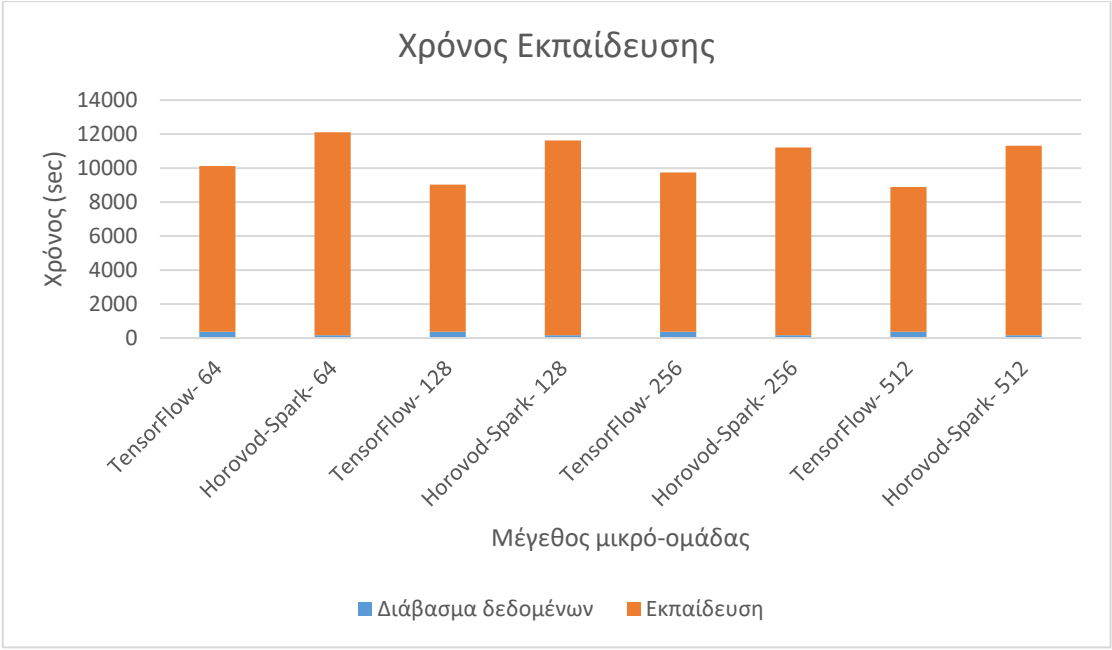
	96	**192**	**384**	**768**	**1536**
Spark / TensorFlow	1,30	2,14	2,07	2,29	2,01

5.2 Αποτελέσματα Cifar-10

5.2.1 Πρώτη Φάση Πειράματος

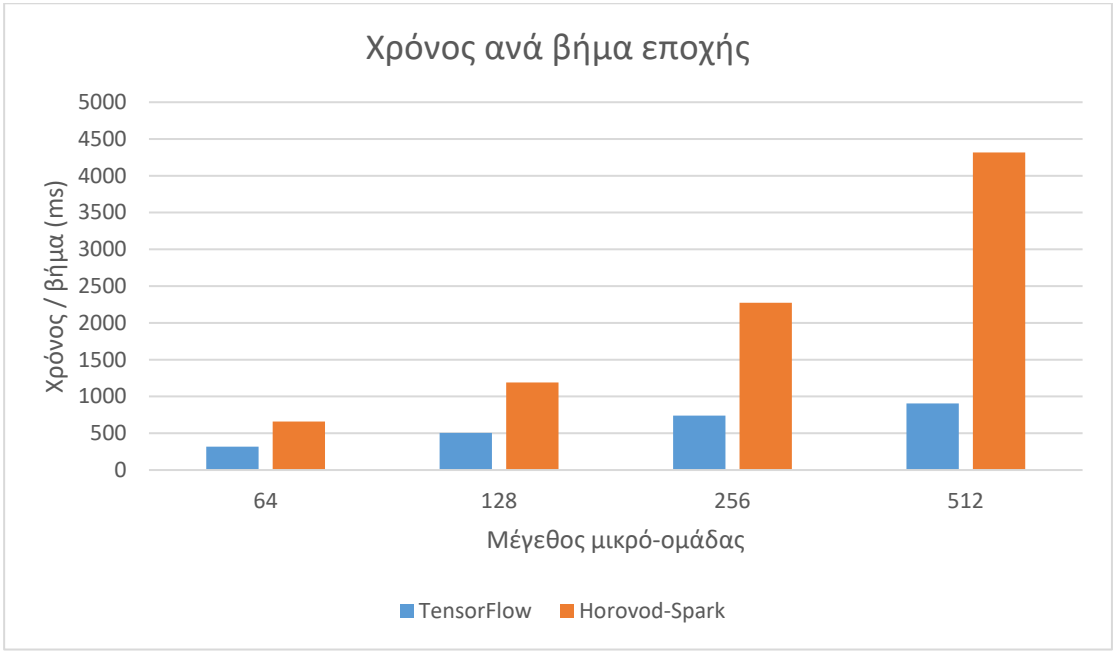
Κατά αντιστοιχία, λοιπόν, με την ενότητα 5.1.1, καταγράφηκαν οι χρόνοι που χρειάστηκαν τα δύο συστήματα για την διαδικασία της εκπαίδευσης. Σε αυτό το πείραμα, το **TensorFlow** κατανάλωνε **6 λεπτά και 7 δευτερόλεπτα** ενώ το **Spark 2 λεπτά και 46 δευτερόλεπτα.**

Στην Εικόνα 5.24 φαίνεται ο χρόνος που χρειάστηκε κάθε σύστημα για την εκπαίδευση του νευρωνικού δικτύου για τις διάφορες τιμές μεγέθους μικρό-ομάδας. Στον χρόνο αυτό συμπεριλαμβάνεται και το διάβασμα και η προ-επεξεργασία των δεδομένων. Έπειτα στην Εικόνα 5.25 καταγράφεται κατά μέσο όρο ο χρόνος που καταναλώνει κάθε σύστημα για την εκπαίδευση μιας μικρό-ομάδας (mini batch), ο χρόνος ενός βήματος δηλαδή μέσα σε μια εποχή. Σημειώνεται ότι σε αυτό το πείραμα τα μεγέθη μικρό-ομάδας κυμαίνονται από 64 έως 512.

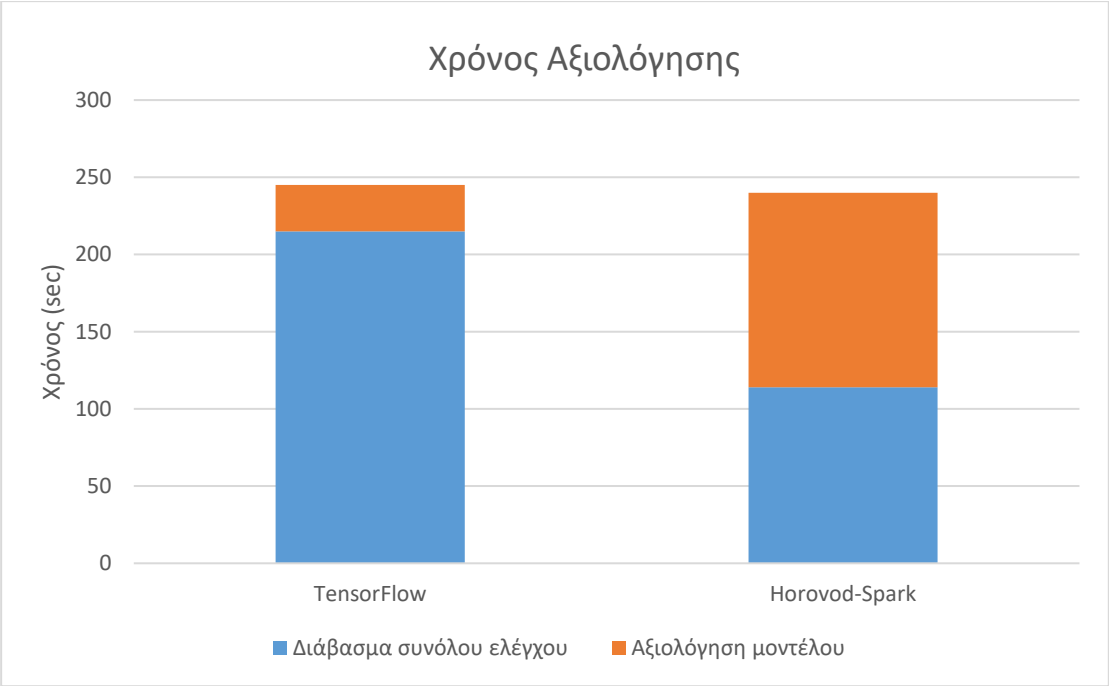


Εικόνα 5.24: Cifar-10 - Διάγραμμα συνολικού χρόνου εκπαίδευσης

Στην συνέχεια ακολουθούν το διάγραμμα χρόνου εκτέλεσης ανά βήμα εποχής και το διάγραμμα συνολικού χρόνου αξιολόγησης.



Εικόνα 5.25: Cifar-10 - Διάγραμμα χρόνου εκτέλεσης βήματος εκπαίδευσης



Εικόνα 5.26 Cifar-10 - Διάγραμμα χρόνου διαβάσματος και αξιολόγησης

Συνολικά, όπως φαίνεται από το σύνολα εκπαίδευσης και ελέγχου και στα δύο πειράματα, φαίνεται ότι το Spark επιδεικνύει καλύτερη συμπεριφορά στο διάβασμα και προ-επεξεργασία των δεδομένων. Αυτό οφείλεται, κυρίως, στην γενική του λειτουργικότητα ως κατανεμημένο σύστημα γενικού σκοπού που στοχεύει σε μεγάλα δεδομένα αλλά και ενδεχομένως η μορφή που ήταν αποθηκευμένα τα σύνολα δεδομένων στο HDFS.

Συνεχίζοντας, περνάμε στους πίνακες που αφορούν την μνήμη που καταναλώνεται και τα ποσοστά χρήσης της CPU. Στο πρώτο πείραμα, όσον αφορά την μνήμη, δεν υπήρχαν ουσιαστικές αυξομειώσεις με την αλλαγή μεγέθους μικρό-ομάδας, όμως εδώ που τα δεδομένα έχουν μεγαλύτερο όγκο και το νευρωνικό δίκτυο είναι αρκετά πιο περίπλοκο και βαρύ, η μνήμη που χρειάζονται τα συστήματα σταδιακά αυξάνεται.

Πίνακας 5.15: Cifar-10 - Μνήμη συστήματος TensorFlow

	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
TensorFlow	64	7 GB	2.4 GB	2.4 GB
	128	9 GB	3 GB	2.8 GB
	256	9.5 GB	3 GB	3 GB
	512	13 GB	4.5 GB	4.5 GB

Πίνακας 5.16 Cifar-10 - Μνήμη συστήματος Spark

	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
Spark	64	14.5 GB	6 GB	4 GB
	128	15 GB	6 GB	4.5 GB
	256	16 GB	6.5 GB	4.5 GB
	512	17 GB	7 GB	5 GB

Όσον αφορά το ποσοστό χρήσης της CPU, φαίνεται ότι στο δεύτερο πείραμα με τα πιο βαριά δεδομένα, το TensorFlow ακόμα και για μικρά μεγέθη μικρό-ομάδας διατηρεί καλό ποσοστό χρήσης, ενώ στο Spark αυξάνεται πολύ λίγο το ποσοστό σε σχέση με πριν.

Πίνακας 5.17 Cifar-10 - Χρήση CPU κατά την εκπαίδευση στο TensorFlow

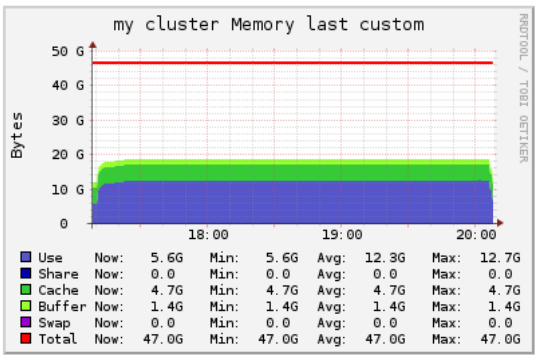
	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
TensorFlow	64	80%	80%	80%
	128	85%	87.5%	87.5%
	256	85%	90%	82.5%
	512	90%	92.5%	90%

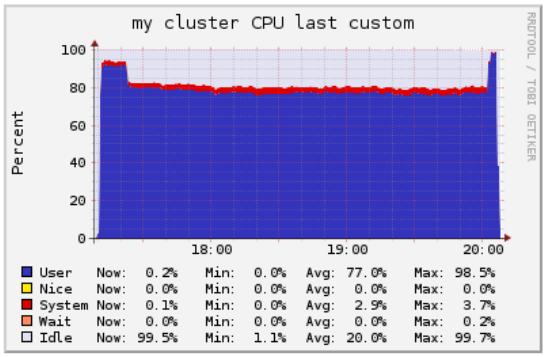
Πίνακας 5.18 Cifar-10 - Χρήση CPU κατά την εκπαίδευση στο Spark

	Μέγεθος μικρό-ομάδας	**Cluster**	**Master**	**Slave**
Spark	64	80%	80%	80%
	128	85%	85%	85%
	256	87.5%	87.5%	87.5%
	512	90%	90%	90%

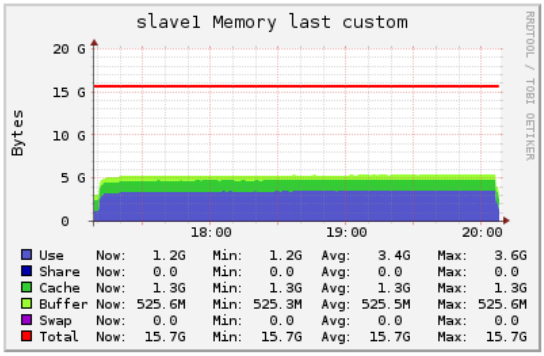
Στην συνέχεια παρουσιάζονται μερικές εικόνες από το Ganglia σχετικά με την μνήμη και την CPU για τις διάφορες τιμές μικρό-ομάδας στα συστήματα TensorFlow και Spark.

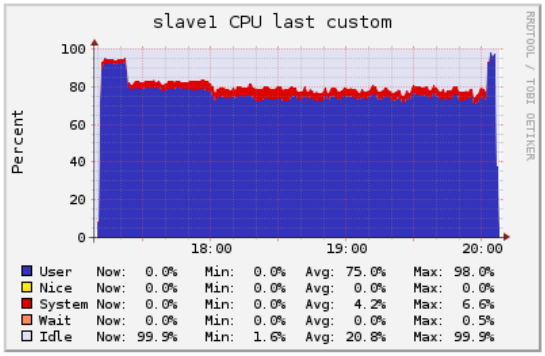
A) TensorFlow



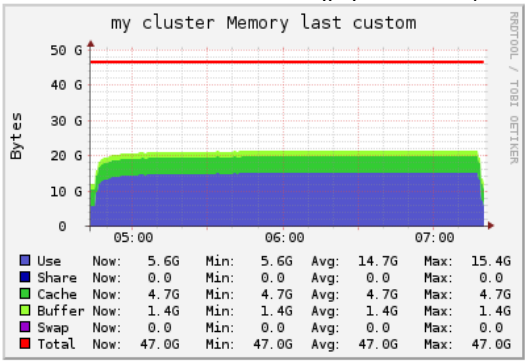


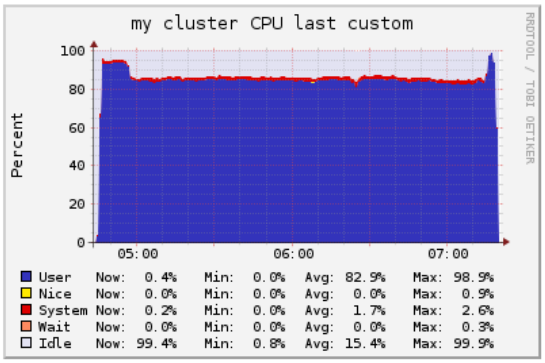
Εικόνα 5.27 Cifar-10 - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 64**



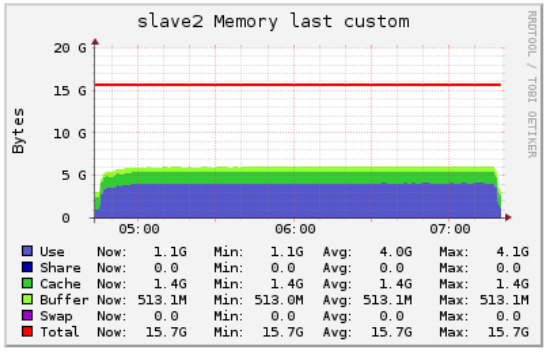


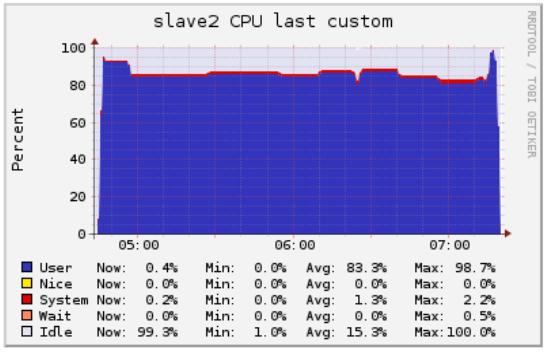
Εικόνα 5.28 Cifar-10 - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 64**



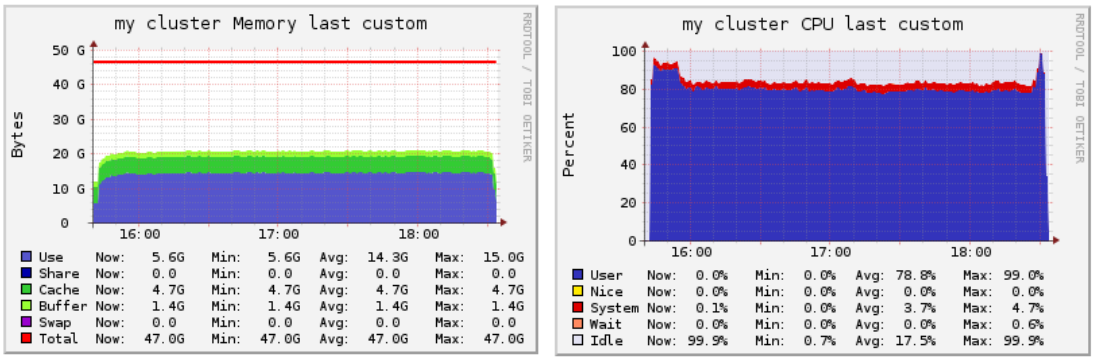


Εικόνα 5.29 Cifar-10 - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 128**

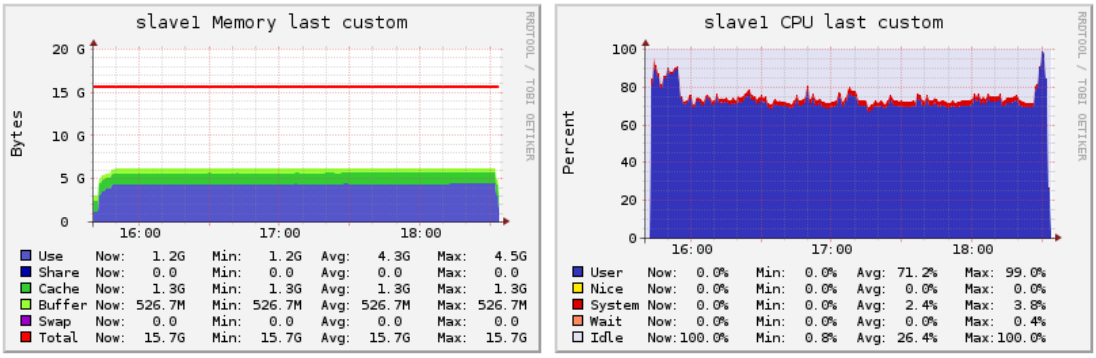




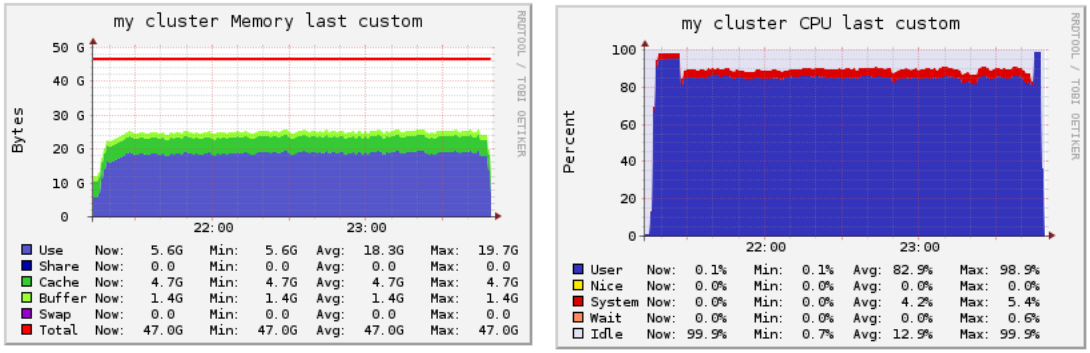
Εικόνα 5.30 Cifar-10 - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 128**



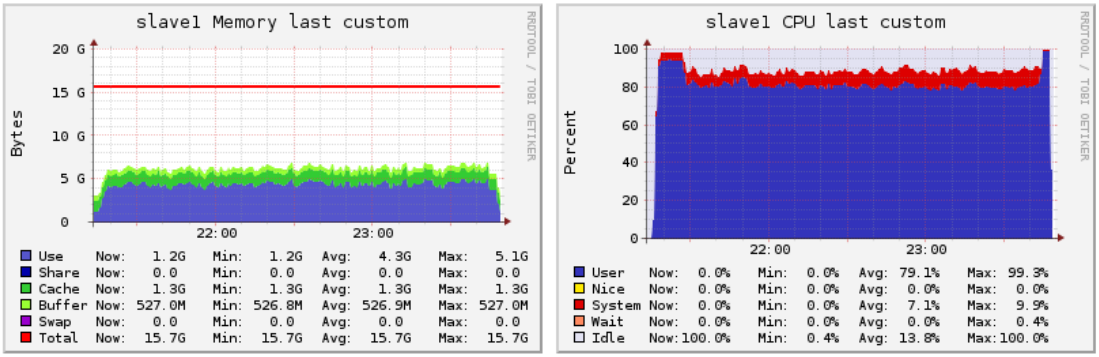
Εικόνα 5.31 Cifar-10 - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 256**



Εικόνα 5.32 Cifar-10 - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 256**

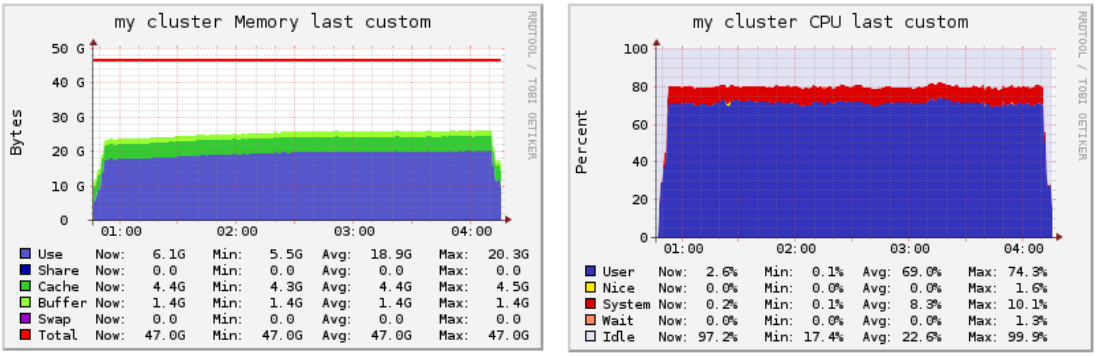


Εικόνα 5.33 Cifar-10 - Μνήμη και CPU για **TensorFlow cluster** με **μέγεθος μικρό-ομάδας 512**

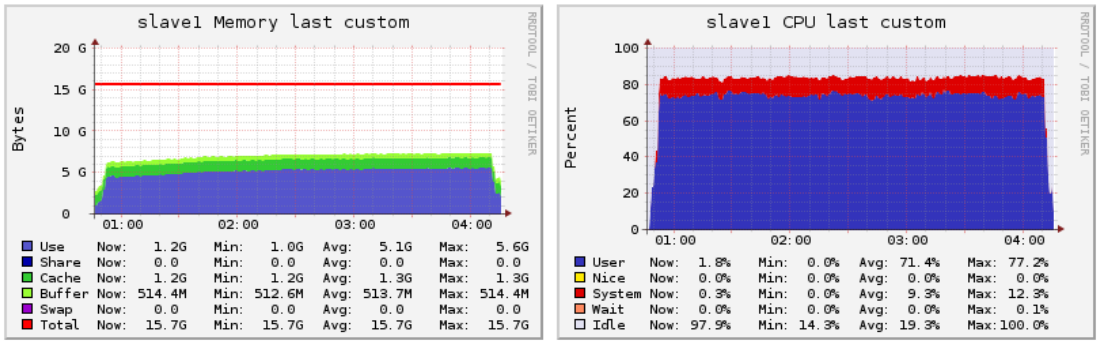


Εικόνα 5.34 Cifar-10 - Μνήμη και CPU για **TensorFlow slave** με **μέγεθος μικρό-ομάδας 512**

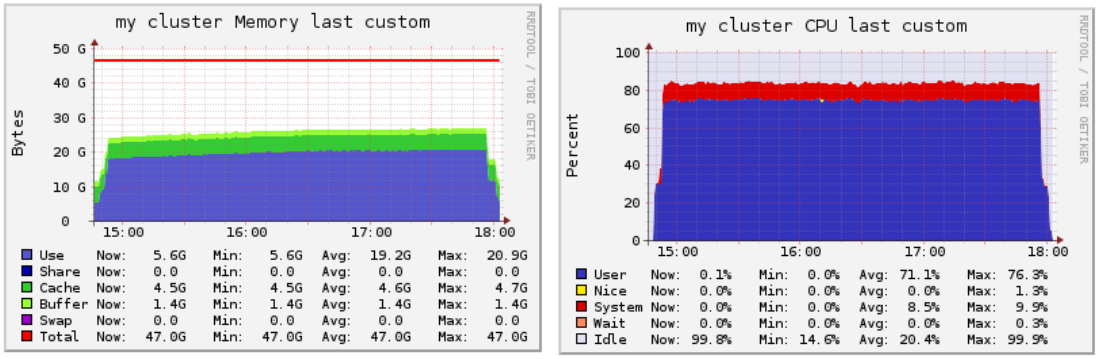
B) Spark



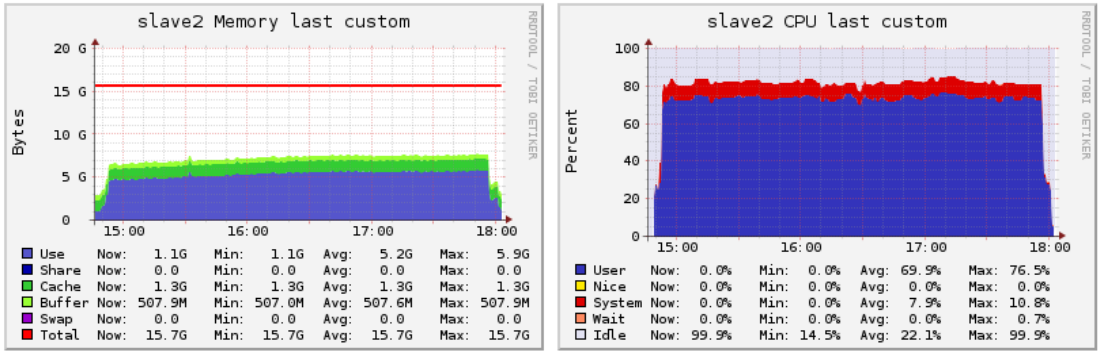
Εικόνα 5.35 Cifar-10 - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 64**



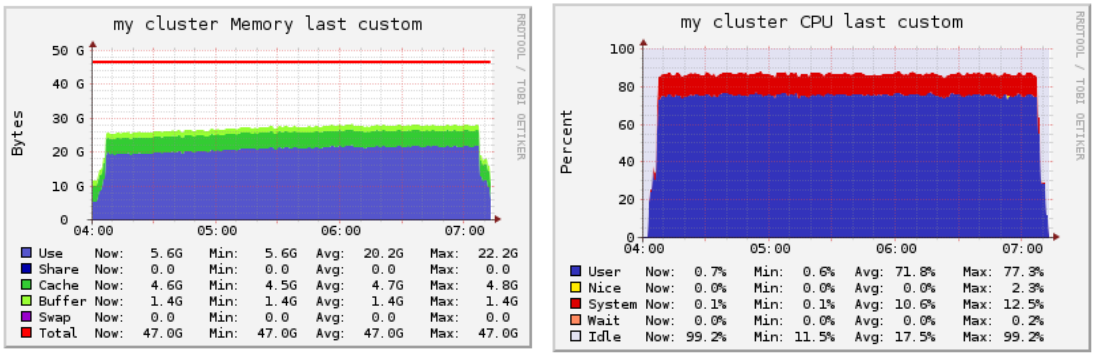
Εικόνα 5.36 Cifar-10 - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 64**



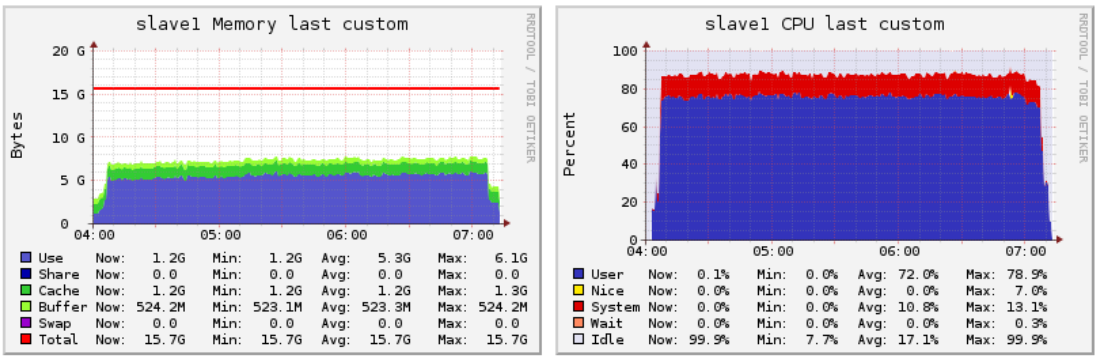
Εικόνα 5.37 Cifar-10 - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 128**



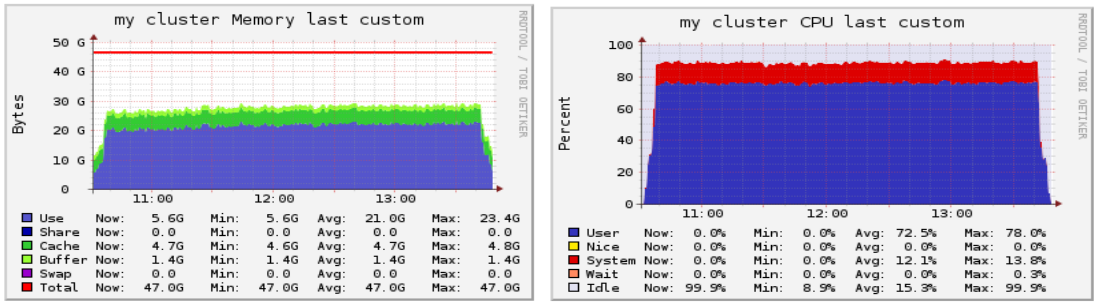
Εικόνα 5.38 Cifar-10 - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 128**



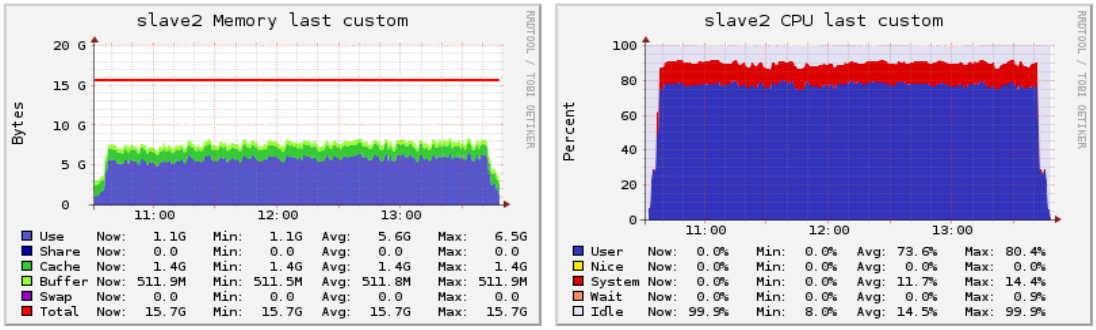
Εικόνα 5.39 Cifar-10 - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 256**



Εικόνα 5.40 Cifar-10 - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 256**



Εικόνα 5.41 Cifar-10 - Μνήμη και CPU για **Spark cluster** με **μέγεθος μικρό-ομάδας 512**



Εικόνα 5.42 Cifar-10 - Μνήμη και CPU για **Spark slave** με **μέγεθος μικρό-ομάδας 512**

Παρόμοια συμπεριφορά με το πρώτο πείραμα δείχνουν να επιδεικνύουν τα δύο συστήματα και συνολικά φαίνεται να μην υπερτερεί αισθητά κάποιο. Όμως αξίζει να σημειωθεί ότι, όπως φαίνεται στις εικόνες 5.27 - 5.42, παρατηρήθηκε μεγαλύτερο ποσοστό χρήσης CPU τύπου "system" στο Spark απ' ότι στο TensorFlow. Συνοπτικά, ο τύπος "user" αναφέρεται σε χρήση της CPU για υπολογισμούς του προγράμματος του χρήστη, συνήθως το μεγαλύτερο ποσοστό είναι σε αυτόν τον

τύπο. Ο τύπος "system" αναφέρεται σε λειτουργίες του πυρήνα (kernel) και περιλαμβάνει συνήθως επικοινωνία προγραμμάτων, εντολές λειτουργικού συστήματος και εντολές εισόδου-εξόδου για αρχεία και διευθύνσεις. Φαίνεται, επομένως, ότι το Spark για το σύνολο cifar-10 δεν αξιοποιεί το ίδιο αποδοτικά την CPU όπως το TensorFlow το οποίο ενδεχομένως αποδίδεται στην επικοινωνία των μηχανημάτων του cluster ή το φόρτωμα δεδομένων από το Store όπου διατηρούνται τα δεδομένα εκπαίδευσης.

5.2.2 Δεύτερη Φάση Πειράματος

Περνώντας στην φάση της σκιαγράφησης (profiling), παρατίθεται πρώτα συνολικά το μέρος των δεδομένων με την μεγαλύτερη επιρροή και στην συνέχεια αναλύονται συγκεκριμένοι operators που εμφανίζουν αξιοσημείωτες διαφορές και σχόλια.

Πίνακας 5.19 Cifar-10 - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 64

	64		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	304534745,00	211458129,67	1,44
Conv2DBackpropInput	172745086,00	120673447,33	1,43
FusedBatchNormGradV3	107371791,33	76129978,00	1,41
_FusedConv2D	79291881,33	63424903,33	1,25
FusedBatchNormV3	50472091,67	41820402,00	1,21
ReluGrad	39194268,67	28962745,67	1,35
BiasAddGrad	22673921,33	12796300,00	1,77
AddN	7672130,67	5763142,33	1,33
Relu	5339324,33	4086452,67	1,31
AddV2	4545005,00	3559743,33	1,28
Mul	6552710,33	2651255,00	2,47
Square	2780928,33	2697331,00	1,03
ResourceApplyAdadelta	15243482,33	9304420,33	1,64
Sum	1362274,00	1434444,33	0,95
AvgPoolGrad	383928,00	369989,00	1,04
Dataset	1767676,67	306476,33	5,77
PyFunc	1068227,33	-	-
HorovodAllreduce/CollectiveReduce	3527170,00	120036433,33	0,03

Πίνακας 5.20 Cifar-10 - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 128

	128		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	286902769,00	227202829,67	1,26
Conv2DBackpropInput	157616715,00	117883207,33	1,34
FusedBatchNormGradV3	93447533,33	73736245,00	1,27
_FusedConv2D	72502314,33	56508434,00	1,28
FusedBatchNormV3	46489861,67	38642460,67	1,20
ReluGrad	33945740,00	28157749,00	1,21
BiasAddGrad	18080633,33	11926962,33	1,52
AddN	6111566,67	5299701,00	1,15
Relu	4331528,33	3388156,33	1,28
AddV2	4248851,67	3504746,00	1,21
Mul	4545121,33	1901870,33	2,39
Square	2273464,00	1922324,67	1,18
ResourceApplyAdadelta	9852331,67	4908486,00	2,01
Sum	1182324,00	1046802,00	1,13
AvgPoolGrad	385958,67	294484,00	1,31
Dataset	1882658,67	180608,67	10,42
PyFunc	1245577,00	-	-
HorovodAllreduce/CollectiveReduce	1898999,33	96792669,33	0,02

Πίνακας 5.21 Cifar-10 - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 256

	256		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	288534623,67	263523621,33	1,09
Conv2DBackpropInput	143972559,33	124380216,00	1,16
FusedBatchNormGradV3	93460204,33	84428423,00	1,11
_FusedConv2D	83328508,33	67960255,00	1,23
FusedBatchNormV3	54698102,00	43880940,67	1,25
ReluGrad	32200057,67	29764769,67	1,08
BiasAddGrad	17681006,00	14210401,00	1,24
AddN	6071030,00	6094275,33	1,00
Relu	4376346,33	3814872,33	1,15
AddV2	4330872,67	4168999,33	1,04
Mul	2551563,00	1419169,67	1,80
Square	1136311,33	1504188,33	0,76
ResourceApplyAdadelta	5819672,67	2432287,00	2,39
Sum	706039,00	1006101,67	0,70
AvgPoolGrad	548837,33	340130,33	1,61
Dataset	1552532,33	69501,33	22,34
PyFunc	1007693,67	-	-
HorovodAllreduce/CollectiveReduce	1249332,67	83079902,00	0,02

Πίνακας 5.22 Cifar-10 - Σύγκριση τελεστών για μέγεθος μικρό-ομάδας 512

	512		
Operator	**Χρόνος Spark (μs)**	**Χρόνος TensorFlow (μs)**	**Spark/ TensorFlow**
Conv2DBackpropFilter	299454931,67	291535542,50	1,03
Conv2DBackpropInput	141506477,67	129554180,50	1,09
FusedBatchNormGradV3	95798158,67	89872088,00	1,07
_FusedConv2D	69375759,67	65286700,00	1,06
FusedBatchNormV3	58127995,00	53672775,00	1,08
ReluGrad	27143737,33	25538265,50	1,06
BiasAddGrad	16459741,00	15190344,50	1,08
AddN	5950886,33	6247571,50	0,95
Relu	4476512,00	4444259,50	1,01
AddV2	4342893,33	4405325,00	0,99
Mul	1187163,00	1481430,00	0,80
Square	286255,67	1449872,00	0,20
ResourceApplyAdadelta	3529449,33	1283037,00	2,75
Sum	132465,33	1099414,50	0,12
AvgPoolGrad	662508,00	648601,50	1,02
Dataset	1561983,33	64605,50	24,18
PyFunc	1050574,33	-	-
HorovodAllreduce/CollectiveReduce	667670,00	7665624,50	0,09

Dataset

Αντίστοιχα με πριν, έχουμε αρχικά τα πρώτα δεδομένα σχετικά με αυτήν την κατηγορία τελεστών στον Πίνακα 5.21.

Πίνακας 5.23 Cifar-10 - Λόγος χρόνου Spark προς TensorFlow για τον operator Dataset

	64	**128**	**256**	**512**
Spark / TensorFlow	5,77	10,42	22,34	24,18

Η σύγκριση των χρόνων **Dataset – Prefetch_Generator** (για TensorFlow) και **Dataset – PyFunc** (για Spark) με σκοπό να γίνει πιο καθαρή η συνεισφορά του μηχανισμού Prefetch φαίνεται στον Πίνακα 5.22.

Πίνακας 5.24: Cifar-10 - Λόγος χρόνου Spark προς TensorFlow για τον operator Dataset ειδικεύοντας για Prefetch

	64	**128**	**256**	**512**
Spark / TensorFlow	2,65	4,06	9,32	8,80

PyFunc

Αυτή η κατηγορία αφορά μόνο το Spark και, όπως αναλύθηκε στην ενότητα 5.1.2, πρόκειται για κάποιους ενδιάμεσους μετασχηματισμούς που χρειάζεται να γίνουν για λόγους συμβατότητας με την βιβλιοθήκη Petastorm που χειρίζεται την σύνδεση των Spark DataFrames και TensorFlow datasets.

Στην συγκεκριμένη περίπτωση, μετρήθηκε ότι προσθέτει κατά μέσο όρο **1093 ms ανά εποχή** επιπλέον φόρτο.

AllReduce

Στην κατηγορία αυτή ανήκουν υπολογισμοί που αφορούν την συλλογή των gradients μέσω του αλγορίθμου AllReduce όπως αναλύθηκε σε προηγούμενο κεφάλαιο. Αρχικά έχουμε τα εξής δεδομένα για τα διάφορα μεγέθη μικρό-ομάδας:

Πίνακας 5.25: Cifar-10 - Λόγος TensorFlow προς Spark για τον operator AllReduce

	64	**128**	**256**	**512**
Spark / TensorFlow	0,03	0,02	0,02	0,09

Κάνοντας, όμως, περισσότερη ανάλυση στους επιμέρους operators αυτής της κατηγορίας, προέκυψαν τα παρακάτω σχόλια και ενημερώσεις της σύγκρισης.

Όσον αφορά το TensorFlow, οι 4 operators :

- div_no_nan/allreduce/CollectiveReduce
- div_no_nan/allreduce_1/CollectiveReduce
- div_no_nan_1/allreduce/CollectiveReduce
- div_no_nan_1/allreduce_1/CollectiveReduce

παρατηρήθηκε ότι εκτελούνται σε ξεχωριστά νήματα, παράλληλα με τους άλλους operators, ενδιάμεσα στην εκπαίδευση της μικρό-ομάδας και δεν προσδίδουν επιπλέον χρονική επιβάρυνση. Όμως, ο operator **scoped_allocator_1_1_CollectiveReduce** εκτελείται στο τέλος, αφού τελειώσουν όλοι οι υπολογισμοί και πριν αρχίσει η ενημέρωση των παραμέτρων με τον βελτιστοποιητή (Adadelta), προσδίδοντας αισθητή χρονική επιβάρυνση καθώς το σύστημα μένει πρακτικά ανενεργό μέχρι να τελειώσει αυτός ο operator για να προχωρήσει στον βελτιστοποιητή. Σημειώνεται επιπλέον ένας συγκεκριμένος operator **allreduce/CollectiveReduce** για μαζικό AllReduce που εκτελείται στην αρχή κάθε εποχής και μετρήθηκε ότι επιβαρύνει περίπου κατά **8 δευτερόλεπτα** αν συνυπολογίσουμε και τον ανενεργό χρόνο (idle time) ή **1,8 δευτερόλεπτα** σε καθαρό χρόνο υπολογισμού.

Από την άλλη μεριά, το Horovod εκτελεί **άμεσα** την διαδικασία AllReduce πάνω σε μεταβλητές που έχουν υπολογιστεί και προχωράει μάλιστα και στην εφαρμογή του βελτιστοποιητή, μέσω της τεχνικής Tensor Fusion που αναφέρθηκε νωρίτερα. Επιπλέον, παρατηρήθηκαν μερικοί operators τύπου AllReduce που ανήκαν στην κατηγορία Mul οπότε μπορούν να προστεθούν και αυτοί στην συνολική σύγκριση. (Σημειώνεται ότι αφαιρώντας αυτούς τους χρόνους από την κατηγορία Mul στο Spark, προέκυψε ότι δεν αξίζει σχολιασμού στην περίπτωσή μας αυτό το κομμάτι, παρά τις μεγάλες τιμές του λόγου στους αρχικούς πίνακες)

Καταλήγουμε, επομένως, σε μια πιο καθαρή παράθεση operators AllReduce των δύο συστημάτων όπου στο TensorFlow υπολογίζουμε μόνο τον scoped_allocator και στο Horovod προσθέτουμε τους επιπλέον operators κάτω από την κατηγορία Mul. Στον Πίνακα 5.26 φαίνεται η τελική σύγκριση.

Πίνακας 5.26: Cifar-10 - Ενημερωμένος λόγος TensorFlow προς Spark για τον operator AllReduce

	64	**128**	**256**	**512**
Spark / TensorFlow	0,14	0,12	0,10	0,33

ResourceApplyAdadelta

Αναφέρεται συνοπτικά και η συγκεκριμένη κατηγορία operators καθώς αφορά τον βελτιστοποιητή Adadelta και φαίνεται ότι το Spark υστερεί όσον αφορά τον υπολογισμό και την ενημέρωση των βαρών.

Πίνακας 5.27: Cifar-10 - Λόγος Spark προς TensorFlow για τον operator ResourceApplyAdadelta

	64	**128**	**256**	**512**
Spark / TensorFlow	1,64	2,01	2,39	2,75

Mul/Sum/Square

Εδώ αξίζει να αναφερθούν 3 συγκεκριμένοι operators:

- conv2d/kernel/Regularizer/Square,
- conv2d/kernel/Regularizer/Sum
- gradient_tape/conv2d/kernel/Regularizer/Mul_1

οι οποίοι παρατηρήθηκε ότι παρουσιάζουν αισθητή διαφορά στο Spark αλλά κυρίως για **μέγεθος μικρό-ομάδας 512.**

Πίνακας 5.28 Cifar-10 - Λόγος Spark προς TensorFlow για operators Mul/Sum/Square για μέγεθος μικρό-ομάδας 512

	Mul	**Square**	**Sum**
Spark / TensorFlow	0,28	0,20	0,12

Αυτοί οι operators συνδέονται με μία παράμετρο των συνελικτικών επιπέδων που χρησιμοποιήθηκαν στο νευρωνικό δίκτυο και ονομάζεται **κανονικοποιητής L2 (regularizer)** και εφαρμόζει μία «ποινή» στον πυρήνα (kernel) του συνελικτικού επιπέδου**.** Ο τύπος υπολογισμού όπως αναφέρεται [67] είναι:

$$Loss = l2 * reduce\_sum(square(x))$$

οπότε γίνεται σαφές που αναφέρονται αυτοί οι υπολογισμοί στο συγκεκριμένο νευρωνικό δίκτυο. Ο κύριος λόγος για αυτήν την αισθητά καλύτερη συμπεριφορά του Spark σε σχέση με το TensorFlow σε αυτούς τους operators και μάλιστα για μεγάλο μέγεθος μικρό-ομάδας είναι η αποτελεσματικότητα και ταχύτητα που παρέχει από μόνο του το Spark σε υπολογισμούς τύπου map / reduce.

Κεφάλαιο 6

Συμπεράσματα και Επεκτάσεις

Στην διπλωματική εργασία αυτή μελετήθηκε η συμπεριφορά των συστημάτων TensorFlow από την μία και Horovod πάνω σε Spark με την βοήθεια της βιβλιοθήκης Keras από την άλλη, σχετικά με την κατανεμημένη εκπαίδευση νευρωνικών δικτύων. Κύριος σκοπός ήταν η σύγκριση αυτών των δύο δομών και η κατανόηση του κατά πόσο ο συνδυασμός ενός συστήματος γενικού σκοπού όπως το Spark και ενός συστήματος βαθιάς μάθησης προσφέρει πλεονεκτήματα.

Αρχικά, στο κομμάτι του διαβάσματος και προ-επεξεργασίας των δεδομένων εκπαίδευσης, ένα βασικό σχόλιο είναι ο επιπλέον φόρτος που υφίσταται η δομή του Horovod πάνω στο Spark για να γίνει μετατροπή από Spark Dataframe σε μορφή Parquet για λόγους συμβατότητας. Παρ' όλα αυτά, παρατηρήθηκε ότι στο σύνολο mnist δεν υπήρχαν αισθητές διαφορές αλλά στο σύνολο cifar-10 το Spark αποδείχθηκε αισθητά πιο αποδοτικό. Σημειώνεται ότι στο TensorFlow αυτό το στάδιο ενσωματώνεται στην διάρκεια της πρώτης εποχής που είναι πάντα πιο αργή μέχρι να παίξει μετά ρόλο η cache. Αυτό μπορεί, μεν, να μην επηρεάζει ουσιαστικά το συνολικό αποτέλεσμα όσον αφορά τον χρόνο αλλά προσδίδει, δε, κάτι θετικό στο TensorFlow καθώς αρχίζει η διαδικασία της εκπαίδευσης με το που γίνει διαθέσιμη η πρώτη μικρό-ομάδα δεδομένων εκπαίδευσης.

Περνώντας τώρα στην φάση της εκπαίδευσης, συνολικά το TensorFlow από μόνο του με την κατανεμημένη υλοποίηση αποδείχθηκε πιο αποδοτικό από το Horovod πάνω σε Spark. Από τα διαγράμματα χρόνου φαίνεται ότι κατά μέσο όρο το TensorFlow χρειάζεται λιγότερο χρόνο για μία εποχή και αυτό αντικατοπτρίζεται εν τέλει και στον συνολικό χρόνο εκπαίδευσης, όπου μάλιστα η διαφορά γίνεται ακόμα πιο μεγάλη περνώντας από το σύνολο mnist στο πιο βαρύ σύνολο cifar-10.

Με την σκιαγράφηση, η πιο καίρια διαφορά που μετρήθηκε ήταν η συνεισφορά της λειτουργικότητας του Prefetch για το TensorFlow όπου το επόμενο βήμα εκπαίδευσης προετοιμάζεται όσο τελειώνει το προηγούμενο, κάτι το οποίο το Spark δεν διαθέτει. Στα μείον του Horovod με Spark επίσης παρουσιάζει ενδιαφέρον η μικρή επιβάρυνση που προκαλείται από την ανάγκη συμβατότητας μεταξύ του Spark και της βιβλιοθήκης Petastorm, κάτι που υποδεικνύει ότι στο κομμάτι που αφορά τον συνδυασμό του Spark και μιας βιβλιοθήκης του TensorFlow (Keras) για τον σκοπό της βαθιάς εκπαίδευσης νευρωνικών δικτύων, υπάρχουν περιθώρια βελτίωσης. Στον αντίποδα, μεγάλο ενδιαφέρον παρουσιάζεται στο κομμάτι του AllReduce που είναι πολύ βασικό κομμάτι της κατανεμημένης εκπαίδευσης. Τα αποτελέσματα της σκιαγράφησης έδειξαν ότι το Horovod επιτυγχάνει πολύ αποδοτικότερη υλοποίηση του AllReduce για τα διανύσματα κλίσης απ' ότι το κατανεμημένο TensorFlow. Τέλος, αξίζει να σημειωθεί το προτέρημα που έχει το Spark σε υπολογισμούς τύπου map / reduce όπως φάνηκε στον υπολογισμό του L2 regularizer.

Τα συμπεράσματα, λοιπόν, θα μπορούσαν να συνοψιστούν στα εξής:

- το κατανεμημένο TensorFlow αποδεικνύεται συνολικά πιο γρήγορο και αποδοτικό, σε μεγάλο βαθμό χάρη στην δυνατότητα του μηχανισμού Prefetch και γενικότερα στον χειρισμό του input pipeline σαν τροφοδότηση του νευρωνικού δικτύου, φτάνοντας έως και 164X ταχύτερο στην προετοιμασία των μικρό-ομάδων σε κάθε βήμα.

- σημαντικό αποτέλεσμα ήταν το γεγονός ότι το Horovod επιτύγχανε αποδοτικότερη υλοποίηση AllReduce με την χρήση του MPI και την τεχνική Tensor Fusion, φτάνοντας έως και 23X ταχύτερη υλοποίηση.

- αξίζει αναφοράς και η μικρή επιβάρυνση που προσδίδει η βιβλιοθήκη Petastorm προκειμένου να επιτευχθεί η μετατροπή Spark Dataframe σε συμβατή μορφή Dataset για το Keras του TensorFlow, κάτι που έχει περιθώρια βελτίωσης για την σύνδεση των δύο διαφορετικών αυτών συστημάτων.

- τέλος, να σημειωθεί ότι η χρήση του Spark μπορεί να προσφέρει ευελιξία και πολλαπλές δυνατότητες όπως για παράδειγμα αξιοποίηση ροών δεδομένων (data streams) σε πραγματικό χρόνο ή ταχύτερη προ-επεξεργασία σε πολύ ογκώδη δεδομένα.

Επεκτάσεις της παρούσας διπλωματικής εργασίας θα μπορούσαν να είναι οι εξής:

- συγκριτική μελέτη για μεγαλύτερα σύνολα εκπαίδευσης και πιο πολύπλοκες αρχιτεκτονικές προκειμένου να φανεί πιο καθαρά η απόδοση των δύο συστημάτων στην υλοποίηση AllReduce.

- διερεύνηση για βελτίωση της σύνδεσης των δομών του Spark και του TensorFlow για αποδοτικότερο συνδυασμό και μείωση του επιπλέον φόρτου αυτής της σύνδεσης

- συγκριτική μελέτη κάνοντας χρήση του PyTorch αντί του TensorFlow για τον συνδυασμό ενός συστήματος γενικού σκοπού τύπου Spark και ενός εξειδικευμένου για βαθιά μηχανική μάθηση

Βιβλιογραφία

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[2] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 6645–6649, doi: 10.1109/ICASSP.2013.6638947.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016, pp. 770–778, Accessed: Feb. 07, 2021. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[4] M. Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2015.

[5] T. Chen *et al.*, "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," p. 6.

[6] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," p. 12.

[7] M. Li *et al.*, "Scaling Distributed Machine Learning with the Parameter Server," 2014, pp. 583–598, Accessed: Feb. 07, 2021. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu.

[8] J. Dean *et al.*, "Large Scale Distributed Deep Networks," 2012.

[9] K. S. Chahal, M. S. Grover, K. Dey, and R. R. Shah, "A Hitchhiker's Guide On Distributed Training Of Deep Neural Networks," *J. Parallel Distrib. Comput.*, vol. 137, pp. 65–76, Mar. 2020, doi: 10.1016/j.jpdc.2019.10.004.

[10] A. Thusoo *et al.*, "Hive - a petabyte scale data warehouse using Hadoop," in *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, Mar. 2010, pp. 996–1005, doi: 10.1109/ICDE.2010.5447738.

[11] M. Kornacker, "Impala: A Modern, Open-Source SQL Engine for Hadoop," p. 34.

[12] R. Sethi *et al.*, "Presto: SQL on Everything," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, Apr. 2019, pp. 1802–1813, doi: 10.1109/ICDE.2019.00196.

[13] "GraphLab," *Wikipedia*. Jan. 02, 2020, Accessed: Mar. 06, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=GraphLab&oldid=933640095.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," p. 7.

[15] "MLlib | Apache Spark." https://spark.apache.org/mllib/ (accessed Mar. 06, 2021).

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Mar. 2010, pp. 249–256, Accessed: Feb. 07, 2021. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html.

[17] J. Heaton, "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning," *Genet. Program. Evolvable Mach.*, vol. 19, no. 1, pp. 305–307, Jun. 2018, doi: 10.1007/s10710-017-9314-z.

[18] "What deep learning is and isn't," *The Data Scientist*, Apr. 17, 2018. https://thedatascientist.com/what-deep-learning-is-and-isnt/ (accessed Feb. 14, 2021).

[19] S. Marsland, *Machine Learning: An Algorithmic Perspective, Second Edition*. CRC Press, 2015.

[20] I. W. on A. N. Networks, J. Mira, and F. Sandoval, *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995 : Proceedings*. Springer Science & Business Media, 1995.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, Art. no. 7553, May 2015, doi: 10.1038/nature14539.
[22] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," p. 8.
[23] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018, doi: 10.1007/s13244-018-0639-9.
[24] S. K. E, "Swapna K E," *Developers Breach*. https://developersbreach.com/author/swapnake123/ (accessed Feb. 08, 2021).
[25] A. Rusiecki, "Trimmed categorical cross-entropy for deep learning with label noise," *Electron. Lett.*, vol. 55, no. 6, pp. 319–320, Feb. 2019, doi: 10.1049/el.2018.7980.
[26] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv160904747 Cs*, Jun. 2017, Accessed: Feb. 07, 2021. [Online]. Available: http://arxiv.org/abs/1609.04747.
[27] "Single-Layer Neural Networks and Gradient Descent," *Dr. Sebastian Raschka*, Mar. 24, 2015. https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html (accessed Feb. 14, 2021).
[28] L. Bottou and T. B. Laboratories, "Stochastic Gradient Learning in Neural Networks," p. 12.
[29] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," p. 39.
[30] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *ArXiv12125701 Cs*, Dec. 2012, Accessed: Feb. 07, 2021. [Online]. Available: http://arxiv.org/abs/1212.5701.
[31] T. K. Leen, T. G. Dietterich, and V. Tresp, *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*. MIT Press, 2001.
[32] R. Zhang and J. Kwok, "Asynchronous Distributed ADMM for Consensus Optimization," in *International Conference on Machine Learning*, Jun. 2014, pp. 1701–1709, Accessed: Feb. 07, 2021. [Online]. Available: http://proceedings.mlr.press/v32/zhange14.html.
[33] M. Li *et al.*, "Parameter Server for Distributed Machine Learning," p. 10.
[34] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005, doi: 10.1177/1094342005051521.
[35] M. Abadi *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," 2016, pp. 265–283, Accessed: Feb. 08, 2021. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.
[36] "Effective TensorFlow 2 | TensorFlow Core." https://www.tensorflow.org/guide/effective_tf2 (accessed Feb. 08, 2021).
[37] "Eager execution | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/guide/eager (accessed Feb. 08, 2021).
[38] "tf.data: Build TensorFlow input pipelines | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/guide/data (accessed Feb. 09, 2021).
[39] "Better performance with the tf.data API | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/guide/data_performance (accessed Feb. 09, 2021).
[40] "Multi-worker training with Keras | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/tutorials/distribute/multi_worker_with_keras (accessed Feb. 09, 2021).
[41] "Distributed training with TensorFlow | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/guide/distributed_training (accessed Feb. 09, 2021).
[42] J. R. Corbin, *The Art of Distributed Applications: Programming Techniques for Remote Procedure Calls*. Springer Science & Business Media, 2012.

[43] M. Zaharia *et al.*, "Apache Spark: a unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016, doi: 10.1145/2934664.
[44] M. Frampton, *Mastering Apache Spark*. Packt Publishing Ltd, 2015.
[45] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.
[46] S. Melnik *et al.*, "Dremel: Interactive Analysis of Web-Scale Datasets," p. 10.
[47] G. Malewicz *et al.*, *Pregel: A system for large-scale graph processing*. 2009.
[48] "Apache Storm." http://storm.apache.org/ (accessed Feb. 10, 2021).
[49] M. R. Ghazi and D. Gangodkar, "Hadoop, MapReduce and HDFS: A Developers Perspective," *Procedia Comput. Sci.*, vol. 48, pp. 45–50, 2015, doi: 10.1016/j.procs.2015.04.108.
[50] "Fig. 1. The Apache Spark Ecosystem.," *ResearchGate*. https://www.researchgate.net/figure/The-Apache-Spark-Ecosystem_fig1_320756907 (accessed Feb. 13, 2021).
[51] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *Int. J. Data Sci. Anal.*, vol. 1, no. 3, pp. 145–164, Nov. 2016, doi: 10.1007/s41060-016-0027-9.
[52] M. Armbrust *et al.*, "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Melbourne Victoria Australia, May 2015, pp. 1383–1394, doi: 10.1145/2723372.2742797.
[53] J. Kroß and H. Krcmar, "Models, Simulations, Measurements, And Analysis For Modeling And Simulating Apache Spark Streaming Applications." Zenodo, Aug. 31, 2016, doi: 10.5281/ZENODO.61243.
[54] X. Meng *et al.*, "MLlib: Machine Learning in Apache Spark," p. 7.
[55] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: a resilient distributed graph system on Spark," in *First International Workshop on Graph Data Management Experiences and Systems*, New York New York, Jun. 2013, pp. 1–6, doi: 10.1145/2484425.2484427.
[56] M. Zaharia *et al.*, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," p. 14.
[57] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *ArXiv180205799 Cs Stat*, Feb. 2018, Accessed: Feb. 11, 2021. [Online]. Available: http://arxiv.org/abs/1802.05799.
[58] A. Gulli and S. Pal, *Deep Learning with Keras*. Packt Publishing Ltd, 2017.
[59] D. Walker and J. Dongarra, "Mpi: A Standard Message Passing Interface," *Supercomputer*, vol. 12, Dec. 1995.
[60] "horovod/horovod," *GitHub*. https://github.com/horovod/horovod (accessed Feb. 12, 2021).
[61] M. Z. Nicanor, "A Comparison between Text, Parquet, and PCAP Formats for Use in Distributed Network Flow Analysis on Hadoop," *J. Adv. Comput. Netw.*, pp. 59–64, 2017, doi: 10.18178/JACN.2017.5.2.241.
[62] M. L. Massie, B. N. Chun, and D. E. Culler, *The Ganglia Distributed Monitoring System: Design, Implementation And Experience*. 2004.
[63] K. Wongsuphasawat *et al.*, "Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 1–12, Jan. 2018, doi: 10.1109/TVCG.2017.2744878.
[64] "TensorFlow Profiler: Profile model performance | TensorBoard," *TensorFlow*. https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras (accessed Feb. 17, 2021).
[65] "Optimize TensorFlow performance using the Profiler | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/guide/profiler (accessed Feb. 17, 2021).

[66] *uber/petastorm*. Uber Open Source, 2021.
[67] "tf.keras.regularizers.L2 | TensorFlow Core v2.4.1," *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2 (accessed Feb. 25, 2021).