NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMPUTER SCIENCE
COMPUTING SYSTEMS LABORATORY

Study and optimization of distributed deep learning under the parameter server architecture

PH.D. DISSERTATION

of

Nikodimos K. Provatas

Athens, July 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Υπολογιστικων Συστηματων

Μελέτη και βελτιστοποίηση κατανεμημένης εκπαίδευσης νευρωνικών δικτύων στην αρχιτεκτονική του εξυπηρετητή παραμέτρων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Νικόδημου Κ. Προβατά

Αθήνα, Ιούλιος 2025.

Εθνικο Μετσοβιο Πολυτεχνειο
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Μελέτη και βελτιστοποίηση κατανεμημένης εκπαίδευσης νευρωνικών δικτύων στην αρχιτεκτονική του εξυπηρετητή παραμέτρων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Νικόδημου Κ. Προβατά

Συμβουλευτική Επιτροπή: Νεκτάριος Κοζύρης
Ιωάννης Κωνσταντίνου
Δημήτριος Τσουμάκος

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 8η Ιουλίου 2025.

.....	.....	.....
Νεκτάριος Κοζύρης	Ιωάννης Κωνσταντίνου	Δημήτριος Τσούμακος
Καθηγητής ΕΜΠ	Αναπλ. Καθήγητης Παν. Θεσσαλίας	Αναπλ. Καθηγητής ΕΜΠ
.....	.....	.....
Γεώργιος Γκούμας	Δημήτριος Γουνόπουλος	Διονύσιος Πνευματικάτος
Καθηγητής ΕΜΠ	Καθηγητής ΕΚΠΑ	Καθηγητής ΕΜΠ

.....
Γεώργιος Πάλλης
Καθηγητής Παν. Κύπρου

Αθήνα, Ιούλιος 2025.

.....
Νικόδημος Κ. Προβατάς
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Νικόδημος Προβατάς, 2025.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.
Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, Καθηγητή Νεκτάριο Κοζύρη, για την καθοδήγηση και εμπιστοσύνη καθ' όλη τη διάρκεια του διδακτορικού μου. Ιδιαίτερες ευχαριστίες οφείλω στον αρχικά μεταδιδακτορικό ερευνητή του εργαστηριου και πλέον καθηγητή Ιωάννη Κωνσταντίνου, με τον οποίο είχα τη στενότερη συνεργασία. Η συμβολή του ήταν καθοριστική στην εξέλιξη της ερευνητικής μου πορείας. Ευχαριστώ τα μέλη του Εργαστηρίου Υπολογιστικών Συστημάτων (CSLab) για το δημιουργικό περιβάλλον και τη συνεργασία όλα αυτά τα χρόνια.

Σε προσωπικό επίπεδο, ευχαριστώ τον πατέρα μου, Κώστα, για την αδιάκοπη στήριξή του. Θα ήθελα επίσης να ευχαριστήσω τις Έφη, Ελένη, Θάλεια, Νίκη και Βίκυ για την παρουσία τους σε όλη τη διάρκεια αυτής της διαδρομής, η καθεμία με το δικό της μοναδικό τρόπο σε διαφορετικές φάσεις αυτή της προσπάθειας και για την ενθάρρυνση όταν τα πράγματα γίνονταν δύσκολα. Ένα ξεχωριστό ευχαριστώ στον Γιώργο, για τη διαρκή υποστήριξη και τη βοήθειά του στη διάρκεια συγγραφης της διατριβής και προετοιμασίας της παρουσίασης, καθώς απότελεσε σημαντικό παράγοντα ώθησης και ενθάρρυνσης στην ολοκλήρωση αυτής της προσπάθειας.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους υποψηφίους διδάκτορες του εργαστηρίου για το όμορφο κλίμα που δημιουργήθηκε κατά τη διάρκεια αυτής εμπειρίας. Ένα πιο προσωπικό ευχαριστώ για τους Εύη και Νίκο, με τους οποίους μοιραστήκαμε αυτήν την εμπειρία από την αρχή, καθώς και για τον Χρήστο, για την αλληλοϋποστήριξη καθόλη τη διάρκεια εκπόνισης της διατριβής.

Στον πατέρα μου και στη
μνήμη της μητέρας μου

Περίληψη

Οι μέθοδοι βαθιάς μάθησης έχουν μεταμορφώσει πλήθος τομέων αξιοποιώντας μεγάλες ποσότητες δεδομένων και πολύπλοκες αρχιτεκτονικές νευρωνικών δικτύων. Παράλληλα οι υπολογιστικές απαιτήσεις των σύγχρονων μοντέλων συχνά υπερβαίνουν τις δυνατότητες ενός μόνο υπολογιστή, οδηγώντας σε ανάγκη για κατανεμημένη εκπαίδευση. Στην παρούσα διατριβή εξετάζουμε την ασύγχρονη εκπαίδευση υπό το πρίσμα της αρχιτεκτονικής του διακομιστή παραμέτρων, με έμφαση στη βελτίωση της απόδοσης και της σταθερότητας. Καταρχήν, μέσω μίας συγκριτική αξιολόγησης στα πλαίσια της διατριβής αποδεικνύεται ότι η αρχιτεκτονική του εξυπηρετητή παραμέτρων προσφέρει σημαντικά υψηλότερη απόδοση στα πλαίσια εκπαίδευσης νευρωνικών δικτύων σε σχέση με γενικού σκοπού αρχιτεκτονικές επεξεργασίας μεγάλων δεδομένων σε κατανεμημένα περιβάλλοντα. Μέσω συστηματικής βιβλιογραφικής ανασκόπησης γίνεται παρουσιάση των ερευνητικών αξόνων που σχετίζονται με την αρχιτεκτονική του εξυπηρετητή παραμέτρων. Ένας από τους πιο κρίσιμους άξονες έρευνας αποτελεί ο έλεγχος της συνέπειας και το πρόβλημα των παλιών διανυσμάτων κλίσης σε χαλαρότερα μοντέλα συγχρονισμού. Για την αντιμετώπιση αυτών, προτείνεται η υβριδική μέθοδος εκπαίδευσης Εναλλαγής - Στρατηγικής (Strategy-Switch), η οποία ξεκινά με σύγχρονη επικοινωνία και στη συνέχεια μεταβαίνει σε ασύγχρονη εκπαίδευση βάσει εμπειρικού κριτηρίου μετάβασης, επιτυγχάνοντας ταχεία σύγκλιση και ακρίβεια μοντέλου. Επιπλέον, προτείνουμε τεχνικές εκ των προτέρων κατανομής δεδομένων που στοχεύουν στην ισορροπία της διανομής των δειγμάτων μεταξύ των εργαζομένων μηχανημάτων που συμμετέχουν στην εκπαίδευση, με αποτέλεσμα τη βελτίωση της συνέπειας της εκπαίδευσης χωρίς την ύπαρξη συγχρονισμού. Τα πειραματικά αποτελέσματα δείχνουν μείωση της μεταβλητότητας στα μετρικά εκπαίδευσης και αξιολόγησης έως 8 φορές και 2 φορές αντίστοιχα σε σύγκριση με την τυχαία κατανομή δεδομένων. Συνολικά, οι προτεινόμενες μέθοδοι ενισχύουν την ασύγχρονη κατανεμημένη εκπαίδευση βαθιάς μάθησης, προσφέροντας πρακτικές λύσεις που συνδυάζουν ταχύτητα και σταθερότητα για πιο κλιμακούμενη και αξιόπιστη εκπαίδευση μεγάλων νευρωνικών δικτύων.

Λέξεις-Κλειδιά: εξυπηρετητής παραμέτρων, βαθιά μηχανική μάθηση, κατανεμημένη εκπαίδευση, ασύγχρονη εκπαίδευση, διαχείριση δεδομένων, ¨μεγάλα' δεδομένα

Abstract

Deep learning has transformed numerous fields by leveraging vast datasets and complex neural architectures, but the computational demands of modern models often exceed single-node capabilities, prompting distributed training solutions. This thesis investigates asynchronous training under the parameter server paradigm, focusing on enhancing both performance and stability. First, a thorough comparative analysis demonstrates that specialized distributed architectures deliver substantially higher throughput than general-purpose data-processing frameworks at large scales. Following a systematic literature review, consistency control and the mitigation of stale gradients as pivotal challenges in asynchronous setups are identified. To address these, a hybrid Strategy-Switch approach is introduced that begins with synchronous communication to identify a promising solution region before transitioning to asynchronous updates based on an empirically derived switching criterion, achieving both rapid convergence and model accuracy. Building on these insights, offline data-sharding techniques are then proposed, designed to preemptively balance sample distributions across workers, thereby reducing gradient variance and improving training consistency. Experimental results show that the proposed data distribution strategies decrease variability in training and validation metrics by up to eightfold and twofold, respectively, compared to random assignment. Collectively, these contributions advance asynchronous distributed deep learning by offering concrete methods to reconcile speed and stability, supporting more scalable and reliable large-scale neural network training.

Keywords: parameter server, deep learning, distributed learning, asynchronous learning, data management, big data

Contents

1 Introduction **33**
1.1 Motivation . . . 33
1.2 Research Direction for the Remainder of the Thesis . . . 36
1.3 Main Contributions . . . 38
1.4 Document Outline . . . 38

2 Distributed Architectures: Quantifying the performance gap between general purpose to machine learning specifics **40**
2.1 Overview of Distributed Architectures . . . 41
2.2 Architecture of Representative Systems . . . 46
2.3 Experimental Evaluation between Map-Reduce and Parameter Server 51
2.4 Key findings and conclusions . . . 59

3 Diving into the Parameter Server: A survey on recent advances **60**
3.1 Parameter server history . . . 61
3.2 Proposed Approaches . . . 63
3.3 Discussion . . . 79
3.4 Focus on Consistency Control . . . 81

4 Synergizing All-Reduce and Asynchronous Parameter Server via Strategy-Switch for Training Efficiency **83**
4.1 Theoretical Background . . . 84
4.2 Benchmarking baseline strategies (network &straggler sensitivity) . 86
4.3 Strategy-Switch: adaptive consistency controller . . . 90
4.4 Evaluation under network/straggler scenarios . . . 94

5 Data Distribution for the Parameter Server **102**
5.1 Why studying data assignment to workers? . . . 102
5.2 Data Modelling and Data Patterns . . . 103
5.3 Mini-batch selection: From single-node training to asynchronous distributed training . . . 107

6 Offline Data Sharding for Stabilizing Parameter Server Training 109
6.1 Data Sharding Techniques 111
6.2 Experimental Setup 115
6.3 Experimental Evaluation 117
6.4 Discussion and Conclusions 123

7 Conclusions and Future Directions 124
7.1 Conclusions 124
7.2 Future Directions 126
7.3 General Research Directions Based on the Thesis Survey 127
Closing Remarks 129

8 Publications 130

A Background on Machine Learning 133

B Locality Sensitive Hashing 135

C Extended Abstract in Greek 137

List of Figures

1.1 Taxonomy of different parallelization strategies in distributed deep learning. Hybrid strategies are also supported. Data Parallelism is highlighted, since it is under the scope of this thesis. 35
1.2 Taxonomy of different communicaton architectures in distributed deep learning. Hybrid strategies are also supported. Parameter server is highlighted, since it is under the scope of this thesis. . . . 35
1.3 Number of publications containing the exact term ("parameter server") or ("data parallel") learning according to Google Scholar search engine. 36

2.1 The Map-Reduce programming model. 42
2.2 Different types of distributed model training. 43
2.3 All-Reduce model training. 44
2.4 Parameter Server Architecture. 45
2.5 TensorFlow Execution Model. 48
2.6 Spark MLlib Execution Model. 50
2.7 Computation time per minibatch in MLlib and TensorFlow (synchronous and asynchronous) on a 141-node cluster. 53
2.8 Cluster CPU usage for Logistic Regression (minibatch=980K) . . . 54
2.9 Reading time per minibatch in MLlib and TensorFlow on a 141-node cluster 55
2.10 Times per minibatch spent for Perceptron in MLlib and TensorFlow on a 141-node cluster 56
2.11 Cluster CPU usage for Perceptron 56
2.12 Logistic and linear regression performance of Spark MLlib, synchronous (Sync TF) and asynchronous TensorFlow (Async TF) on a 5-node cluster 57
2.13 Perceptron performance of Spark MLlib, synchronous (Sync TF) and asynchronous TensorFlow (Async TF) on a 5-node cluster. 58

3.1 Parameter server research areas 64
3.2 Sequence diagram presenting 3 workers training under a BSP (left) and ASP (right) Parameter Server for k global iterations 66

3.3 Sequence diagram presenting 3 workers training under k-batch-BSP (upper left), k-batch-BSP (upper right), k-ASP (down left) and k-batch-ASP (down right) for $k = 2$. 67
3.4 Sequence diagram presenting 3 workers training under different SSP for $s_{thres} = 2$. 68
3.5 Consistency control approaches in the parameter server. 70
3.6 Network related optimizations for the parameter server 70
3.7 Parameter management for the parameter server. Whitesmoke color area (up) presents parameter storage techniques, while the pale orange area (down) indicates load balancing approaches 74
3.8 Straggler handling approaches in the parameter server 76
3.9 Fault tolerance approaches in the parameter server 78

4.1 Distributed training architectures: (a) All-Reduce; (b) Parameter Server. Dashed lines indicate communication, while solid lines indicate data extraction. 85
4.2 Examples of CIFAR-10 images for each category. 86
4.3 #C1 Cluster CPU Utilization for training benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server* 88
4.4 #C1 Cluster Network Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server* 88
4.5 #C2 Cluster CPU Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server* 89
4.6 #C2 Cluster Network Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server* 89
4.7 StrategySwitch-α% 91
4.8 Empirical Rule StrategySwitch 92
4.9 Accuracy vs. time trade-offs in #C1 cluster for each benchmark when using *All-Reduce*, *Parameter Server* and *Strategy-Switch-α%* with various switching points. Percentage in *Strategy-Switch* labels indicates the value of α%. (a) #B1 - Training Acc. (b) #B1 - Validation Acc. (c) #B2 - Training Acc. (d) #B2 - Validation Acc. 95
4.10 Value used in the empirical rule per training epoch on *All-Reduce* setups for both benchmarks. Vertical dashed line indicates the switching point according to the empirical rule. (a) #B1 - Validation Loss (b) #B2 - Validation Loss. 97
4.11 Training time under *All-Reduce*, *Strategy-Switch* and *Parameter Server* in the homogenous #C1 cluster (a) #B1 benchmark (b) #B2 benchmark 98

4.12 Training and validation accuracy values at convergence, when training in the homogenous #C1 cluster for the benchmarks (a) #B1 benchmark (b) #B2 benchmark 98
4.13 Loss and accuracy, when training in the homogenous #C1 cluster for the benchmarks. Black lines indicate training metrics and gray lines indicate validation metrics. (a) Loss - #B1 benchmark (b) Loss - #B2 benchmark (c) Accuracy - #B1 benchmark (d) Accuracy - #B2 benchmark 99
4.14 Training time under *All-Reduce*, *Strategy-Switch* and *Parameter Server* in the heterogeneous #C2 cluster (a) #B1 benchmark (b) #B2 benchmark 100
4.15 Training and validation accuracy values at convergence, when training in the heterogeneous #C2 cluster for the benchmarks (a) #B1 benchmark (b) #B2 benchmark 100
4.16 Loss and accuracy, when training in the heterogeneous #C2 cluster for the benchmarks. Black lines indicate training metrics and gray lines indicate validation metrics. (a) Loss - #B1 benchmark (b) Loss - #B2 benchmark (c) Accuracy - #B1 benchmark (d) Accuracy - #B2 benchmark 101

5.1 Class Population Histogram of Imagenet data obtained from Flickr. 103
5.2 Random (right) versus Stratified (left) Assignment. Workers have access to all classes in stratified assignment, while in random each worker has a different view on the dataset. 106
5.3 Three of CIFAR-100 coarse-grained labels split into the corresponding fine-grained labels. For each fine-grained label two example images are provided. 107

6.1 CNN Architecture 110
6.2 Loss of multiple distributed training runs of a simple CNN network over CIFAR-10 dataset. 110
6.3 Offline Data Splitter 111
6.4 Stratified assignment: Half of the points from each class are assigned to each shard. 112
6.5 Stratified data sharding to workers using mod 113
6.6 Example of 2-dimensional points organized in dense (A, B) and sparse (C, D) neighbourhoods. 113
6.7 Distribution aware data sharding to workers using mod 114
6.8 Box plots representing the number of class examples in each shard created randomly for 3 of the runs. 120
6.9 Average Sharding Technique Time per Dataset 122

7.1 Vision of Data Aware Parameter Server Approach. 127

A.1 Contour Plot outlining Gradient Descent and Mini-Batch Gradient Descent while converging to a local minimum point. 134

List of Tables

1.1 Key Deep Learning Models and Their Characteristics (2015–2025) 34
1.2 Prominent Data-Parallel Parameter-Server Systems (2012–2024) . . 37

2.1 TensorFlow vs. Spark MLlib 47
2.2 Real datasets 52
2.3 Synthetic datasets 53
2.4 Read Throughput of Spark MLlib and TensorFlow 55
2.5 Number of minibatches processed per system for logistic and linear regression to converge 57
2.6 Number of minibatches processed per system for perceptron to converge 59

3.1 Well-known systems adopting the parameter server architecture . . 62
3.2 Consistency Control Algorithms 66
3.3 Surveys on parameter server architecture. 80

4.1 Benchmark Overview 86
4.2 VM Specifications in Clusters 87
4.3 Clusters used for Experiments 87

6.1 Dataset Characteristics 116
6.2 Resnet-56v1 Singlenode Training Configuration 116
6.3 Statistics (5 runs) of final Training and Validation Loss / Accuracy on the CIFAR-10 dataset per method 117
6.4 Sparsely populated neighbourhoods through Distribution Aware Algorithm (various cluster sizes as input) with the resulting validation metrics for 3 of the runs (CIFAR-10). Mean Size refer to the mean population of all those neighbourhoods. 118
6.5 Statistics (5 runs) of final Training and Validation Loss / Accuracy on the Coarse CIFAR-100 dataset per method 119
6.6 Statistics (5 runs) of final Training and Validation Loss / Accuracy on the Fine CIFAR-100 dataset per method 121

Glossary of Technical Terms

Acronym	Meaning
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASP	Asynchronous Parallel (training)
AWS	Amazon Web Services
BERT	Bidirectional Encoder Representations from Transformers
BSP	Bulk Synchronous Parallel
CIFAR-10	Canadian Institute For Advanced Research 10-Class Image Dataset
CIFAR-100	Canadian Institute For Advanced Research 100-Class Image Dataset
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DFS	Distributed File System
DNN	Deep Neural Network
DSH	Data Sensitive Hashing
DSSP	Dynamic Stale Synchronous Parallel
EPS	Elastic Parameter Server
FL	Federated Learning
GD	Gradient Descent
GLUE	General Language Understanding Evaluation benchmark
GMM	Gaussian Mixture Model
GPT	Generative Pre-trained Transformer
GPT-2	Generative Pre-trained Transformer 2

Acronym	Meaning
GPT-3	Generative Pre-trained Transformer 3
GPT-4	Generative Pre-trained Transformer 4
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
HIGGS	Higgs Boson Dataset
IEEE	Institute of Electrical & Electronics Engineers
KV	Key–Value (store)
LDA	Latent Dirichlet Allocation
LLM	Large Language Model
LSH	Locality-Sensitive Hashing
MFU	Memory Footprint Utilisation
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology dataset
MPI	Message Passing Interface
MXNet	Mobile/Multiple-platform eXecution Network
NLP	Natural Language Processing
OSP	Overlap Synchronous Parallel
PCA	Principal Component Analysis
PMLR	Proceedings of Machine Learning Research
PS	Parameter Server
RAM	Random-Access Memory
RDD	Resilient Distributed Dataset
SGD	Stochastic Gradient Descent
SLR	Systematic Literature Review
SQL	Structured Query Language
SSP	Stale Synchronous Parallel
TF	TensorFlow
VM	Virtual Machine

Chapter 1

Introduction

1.1 Motivation

Over the last decade, deep learning has emerged as a highly popular and influential field, primarily due to its success in numerous big data applications. Neural networks have been adopted in image classification [1, 2], speech recognition [3, 4], and natural language processing [5, 6], among other domains [7, 8]. This uptake is largely attributed to the growing availability of massive datasets and ever more powerful computational resources.

Traditional machine learning algorithms often saturate in performance once the available training data surpasses a certain volume [9], whereas deep neural networks consistently demonstrate improved performance with increasing data [10, 11]. As one of the leaders of the Google Brain Project figuratively remarked, *"the analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms"* [12]. This combination of large-scale data and advanced computational power has propelled deep learning to the forefront of many domains.

A compelling example of deep learning's growth trajectory is Microsoft's ResNet architecture [13], introduced in 2015, which achieved a top-1 accuracy of 78% on the ImageNet dataset [14]. More recent models now contain hundreds of millions or even billions of parameters and can reach 90% accuracy or higher on ImageNet [15–19]. A summary of key models over the past decade—including their parameter counts, benchmark performance, and origin—is provided in Table 1.1. In parallel, large-scale networks for natural language processing have also grown dramatically. In the same year ResNet was proposed, Digital Reasoning introduced a 160-billion-parameter network [20] for language tasks. Over the past decade, deep learning models have scaled from architectures with tens of millions of parameters to state-of-the-art systems containing hundreds of billions or even trillions of parameters. GPT-4 is a prominent example of this trend [21, 22]. Notably, the most recent years have witnessed an accelerated proliferation of large language models (LLMs),

Table 1.1: Key Deep Learning Models and Their Characteristics (2015–2025)

Model	Year	Parameters	Performance	Proposed By	Ref.
ResNet-152	2015	∼60M	Top-1: 78.57% (ImageNet)	He et al.	[13]
ResNeXt-101	2017	∼44M	Top-1: 78.8% (ImageNet)	Xie et al.	[23]
BERT-Large	2018	340M	GLUE: 80.5	Devlin et al.	[24]
GPT-2	2019	1.5B	Perplexity: 18.34 (WikiText-2)	Radford et al.	[25]
T5-11B	2019	11B	GLUE: 89.7	Raffel et al.	[26]
ViT-L/16	2020	307M	Top-1: 85.59% (ImageNet)	Dosovitskiy et al.	[27]
GPT-3	2020	175B	LAMBADA: 76.2% (few-shot)	Brown et al.	[21]
LLaMA 65B	2023	65B	Beats GPT-3 on multiple NLP benchmarks	Touvron et al.	[28]
Falcon-180B	2023	180B	Near PaLM-2 Large performance	Almazrouei et al.	[29]
Mistral 7B	2023	7B	Beats LLaMA-13B on reasoning/math/code	Jiang et al.	[30]
GPT-4	2023	∼1.8T (est.)	Bar Exam (top 10%)	OpenAI	[31]
Claude 2	2023	Not disclosed	MMLU: 78.5 (5-shot)	Anthropic	[32]
Gemini 1.5 Pro	2024	Not disclosed	1M token context window	Google DeepMind	[33]
Gemini 2.0 Flash	2025	Not disclosed	2× speed, strong multimodal	Google DeepMind	[34]

including Claude 2, Gemini, Falcon, LLaMA, and Mistral—each exemplifying different strategies in scaling, training regimes, and accessibility (see Table 1.1).

Despite the remarkable performance gains offered by these large models, training them is increasingly compute-intensive. According to Stanford's 2024 AI Index Report [22], Google's Gemini Ultra required around 100 billion petaFLOPS (with an estimated cost of $200M), while OpenAI's GPT-4 required 10 billion petaFLOPS (costing about $80M). Modern hardware accelerators, such as GPUs, have proven essential to make neural network training feasible [35, 36]. Still, as datasets and models continue to expand, single-machine training becomes prohibitively time-consuming, prompting the use of distributed training approaches [37].

To handle this growing complexity, distributed deep learning typically adopts either *model parallelism* [38, 39] or *data parallelism* [40, 41]. In model parallelism, different parts of a network's layers or parameters are spread across multiple machines. In contrast, data parallelism splits the dataset into shards, each assigned to a particular machine, allowing every worker to train the same global model on a disjoint subset of data. Data parallelism has become the most widely used strategy, as it can be applied irrespective of the neural network architecture and

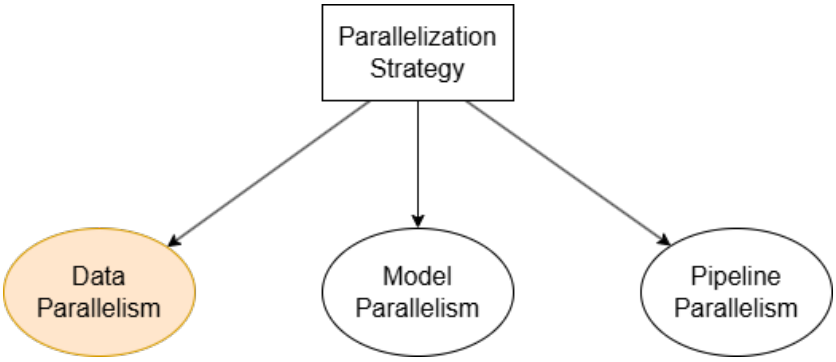


Figure 1.1: Taxonomy of different parallelization strategies in distributed deep learning. Hybrid strategies are also supported. Data Parallelism is highlighted, since it is under the scope of this thesis.

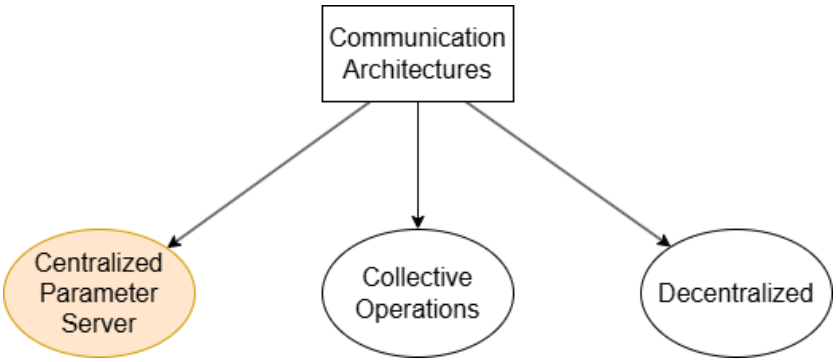


Figure 1.2: Taxonomy of different communicaton architectures in distributed deep learning. Hybrid strategies are also supported. Parameter server is highlighted, since it is under the scope of this thesis.

readily exploits large datasets for performance improvements. Other well-known parallelization strategies include *pipeline parallelism* [42–44], where a pipeline of microbatches is executed across shards of model layers, and hybrid approaches [44], where two or more parallelism strategies can be combined. Different parallelization strategies for distributed deep learning are depicted in Figure 1.1.

Apart from the parallelization strategy, another important aspect of distributed learning is related to the choice of communication architecture, where available options are outlined in Figure 1.2. Worker machines that participate in model training can be organized to communicate through *collective operations* like ring, tree or hierarchical All-Reduce. Another choice includes totally decentralized networks of workers, like peer-to-peer approaches with stochastic or cyclic data exchange with neighbors. Another very well-known and widely reserched and used communication approach is the centralized *parameter server*. In a parameter server setup, multiple

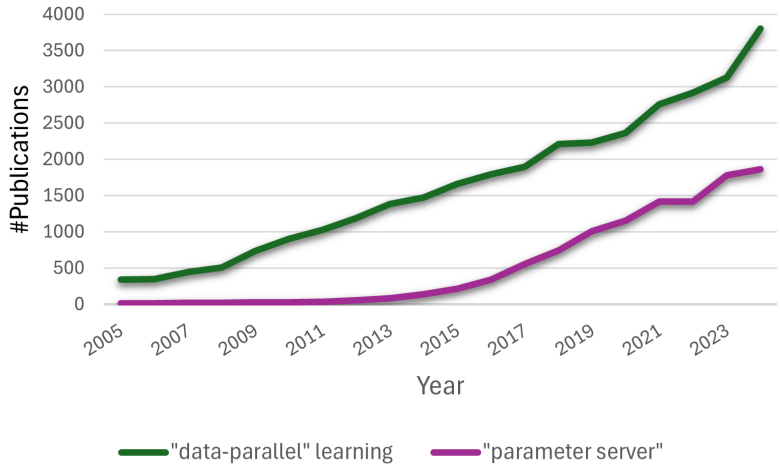


Figure 1.3: Number of publications containing the exact term ("parameter server") or ("data parallel") learning according to Google Scholar search engine.

workers compute gradient updates on distinct data shards and communicate these gradients to centralized parameter servers, which maintain the global model. By focusing on efficient gradient exchange and synchronization, the parameter server approach can handle massive networks and datasets.

From the various approaches that can be adopted for distributed deep learning, this thesis dives into the prime example of the data-parallel *parameter server* [39, 57–59]. It is important to mention that parameter server can also support model parallelism or hybrid parallelization strategies, but these paradigms are out of scope of this thesis. As shown in Figure 1.3, according to Google Scholar searches, research interest in parameter server architectures has risen steadily over the past decade. The same constant increase in research interest is identified while searching about data parallelism, which is expected in an era where data volume is exploding. Parameter servers are also widely used in various deep learning systems over the last decade following various synchronization approaches. Prominent examples are outlined in Table 1.2.

1.2 Research Direction for the Remainder of the Thesis

The thesis starts with examining the need for specialized architectures, like the parameter server, in data parallel setups. Specifically, the thesis evaluates the performance of both the synchronous and asynchronous version of the *Parameter Server* architecture with general-purpose approaches. Having acknowledged the

Table 1.2: Prominent Data-Parallel Parameter-Server Systems (2012–2024)

System	Year	Mode / Feature	Key Contribution	Proposed By	Ref.
DistBelief	2012	Async Downpour-SGD	First large-scale PS; hundreds of CPU workers; template for modern designs	Google	[39]
Petuum / Bösen	2014	SSP consistency	Stale-Synchronous Parallel trades throughput for accuracy guarantees	CMU / Petuum	[45]
TensorFlow PS Strategy	2015	Sync/Async, Elastic	Canonical production PS with fault-tolerant checkpoints	Google	[46]
MXNet KVStore / `ps-lite`	2015	Per-layer Sync/Async	Highly-tuned C++ KV PS usable on CPU or NVLink GPU islands	DMLC	[47]
PaddlePaddle Fleet	2016	Geo-distributed PS	Industrial trainer/PS split; hybrid PS + collective modes	Baidu	[48]
CNTK (legacy)	2016	Block-Momentum PS	Early deep-speech networks before CNTK moved to collectives	Microsoft	[49]
SageMaker Parameter Server	2018	Managed, Elastic	One-flag cloud PS; auto-placement and spot-aware checkpoints	AWS	[50]
BytePS	2019	GPU Relay PS	Overlaps comm/compute; near-linear BERT-Large scaling to 256 GPUs	ByteDance	[51]
MindSpore PS mode	2021	Ascend NPU PS	Trained 200 B-param PanGu-α on 2048 NPUs	Huawei	[52]
Ray PS (actors)	2021	Actor-based PS	$\sim$100 LOC reference; hot-scales across heterogeneous nodes	UC Berkeley / Anyscale	[53]
SageMaker Distributed PS	2022	Elastic, Telemetry	Revamped library with throughput dashboard	AWS	[54]
Elastic PS (EPS)	2022	Runtime rescaling	Grow/shrink PS & workers without restart; hyper-param sweeps	Tencent	[55]
ACK Elastic PS	2024	K8s Spot-tolerant	Adds/removes PS workers on-the-fly for cost-aware training	Alibaba Cloud	[56]

need for such an approach, the thesis dives into existing research related to the parameter server. Asynchronous training typically achieves faster convergence, whereas synchronous approaches provide greater stability. Many intermediate strategies have been proposed to balance these extremes, primarily aiming to mitigate the negative impact of stale gradients by reducing synchronization overhead. This thesis proposes novel techniques specifically designed to leverage asynchronous training, while systematically addressing and minimizing the adverse effects associated with gradient staleness. Specifically, the following question is aimed to be answered by this thesis:

How far can we push the efficiency and stability of large – scale distributed

asynchronous deep-learning in the parameter server architecture?

1.3 Main Contributions

The main contributions of this thesis are the following:

1. The performance gap of the specialized distributed deep learning Parameter Server architecture compared to a widely used distributed processing framework, *e.g.,* MapReduce is evaluated. By comparing representative systems for each approach, namely Google TensorFlow and Apache Spark, an average performance gap of 8.23× when scaling to 140 nodes is observed.

2. A comprehensive systematic literature review (SLR) of research scopes under the parameter server architecture is presented.

3. Strategy-Switch is being introduced. This approach uses All-Reduce to identify a better local minimum to start asynchronous parameter server training.

4. Data distribution strategies that can further benefit parameter server training are discussed.

5. The impact of systematic data sharding approaches rather than random sharding is measured. Using systematic sharding, training and validation metrics exhibit up to 8× and 2× less variance, respectively, across multiple training runs, indicating improved training stability.

1.4 Document Outline

Below is a brief overview of the remaining chapters of this document.

- Chapter 2 discusses MapReduce as a general-purpose architecture in the machine learning context and compares it with the specialized Parameter Server Architecture. It also provides a detailed experimental evaluation of MapReduce versus Parameter Server, highlighting the need for specialized architectures.

- Chapter 3 presents a systematic literature review on the parameter server architecture, focusing on five different areas: consistency control, network optimization, parameter management, straggler problem and fault tolerance.

- Chapter 4 introduces Strategy-Switch, a hybrid communication architecture strategy that uses an empirical rule to initiate asynchronous parameter server training, after some warm-up training with All-Reduce.

- Chapter 5 outlines the vision for exploiting data properties in asynchronous learning, while Chapter 6 measures the effects of sharding data according to their distribution prior to training.
- Chapter 7 presents any ideas for extending this research, and Chapter 8 enumerates the articles published throughout the research that was performed to write this thesis.

Chapter 2

Distributed Architectures: Quantifying the performance gap between general purpose to machine learning specifics

The continuous growth in the volume of available data has prompted researchers to develop new data-parallel system architectures capable of handling big data workloads. Various systems now implement data parallelism under different paradigms. Some of these architectures are designed to be general-purpose, making them suitable for various workloads such as relational, graph, or machine learning tasks. One of the most widely adopted programming models for general big data processing is MapReduce, which is integrated into numerous general-purpose systems, such as Apache Spark [60], which employs libraries like MLlib [61] and BigDL [62] for distributed machine learning.

However, different types of workload often necessitate specialized architectures. In the context of distributed training for machine and deep learning models, multiple solutions have been proposed, with the parameter server emerging as one of the most prominent in well-known deep learning systems, such as TensorFlow [63] and PyTorch [64].

These developments collectively highlight how distributed deep learning — whether implemented in general-purpose engines or specialized frameworks — aims to scale neural network training efficiently to tackle growing model sizes and increasingly large datasets.

In order to understand the benefits from using specialized machine learning architectures compared to general-purpose ones, a detailed experimental evaluation between representative systems is performed in this Chapter. The Chapter begins with an overview on how a general-purpose big data architecture, like MapReduce, can be applied on learning context and then the specialized architecture of param-

eter server is presented. The two distributed architectures are benchmarked on a wide variety of workloads. In particular, Spark's MLlib module, following the general-purpose Map-Reduce architecture, is compared with TensorFlow, which follows the *parameter server* architecture. The choice of this systems is related to the time this experimental evaluation was performed. Specifically, when this evaluation started, TensorFlow was the most popular machine learning library based on job advertisements [65]. On the other hand, Apache Spark was one of the most important tools for data scientists [66] and, in parallel, a general-purpose framework with a very actively contributing community.

The main contribution of the experimental evaluation provided in this Chapter are the following:

- The performance gap between rerpresentative generalized and specialized ML systems is quantified.
- Experiments with both real and synthetic datasets using up to 140 nodes are performed.
- A thorough analysis of the experimental results is performed, focusing on both the architectural and the implementation differences of the systems and major insights are presented.

2.1 Overview of Distributed Architectures

2.1.1 Map-Reduce: Machine Learning Applications

Google first introduced the Map-Reduce programming model in 2004 [67] to facilitate big data processing. Applications that use this model typically consist of two primary functions: *map* and *reduce*. The map function transforms the input data into a set of key-value pairs (either simple or complex), designed according to the specific requirements of the application. These key-value pairs are then passed to the reduce function, which aggregates all values corresponding to the same key.

Figure 2.1 provides an overview of the Map-Reduce programming model. The input data is split into small chunks (commonly 128 MB), which are processed by map tasks that generate key-value pairs as intermediate results. Depending on the implementation, these intermediate results may be stored in memory or written to a file system. Next, each reduce task retrieves data with matching keys and combines their associated values. The final outputs are then written back to disk. Big data solutions based on Map-Reduce typically include a sequence of map and reduce functions that iteratively transform the original dataset into the desired result. Common distributed file systems used in conjunction with Map-Reduce include the Google File System [68], the Apache Hadoop File System [69, 70], and Amazon S3 [71].

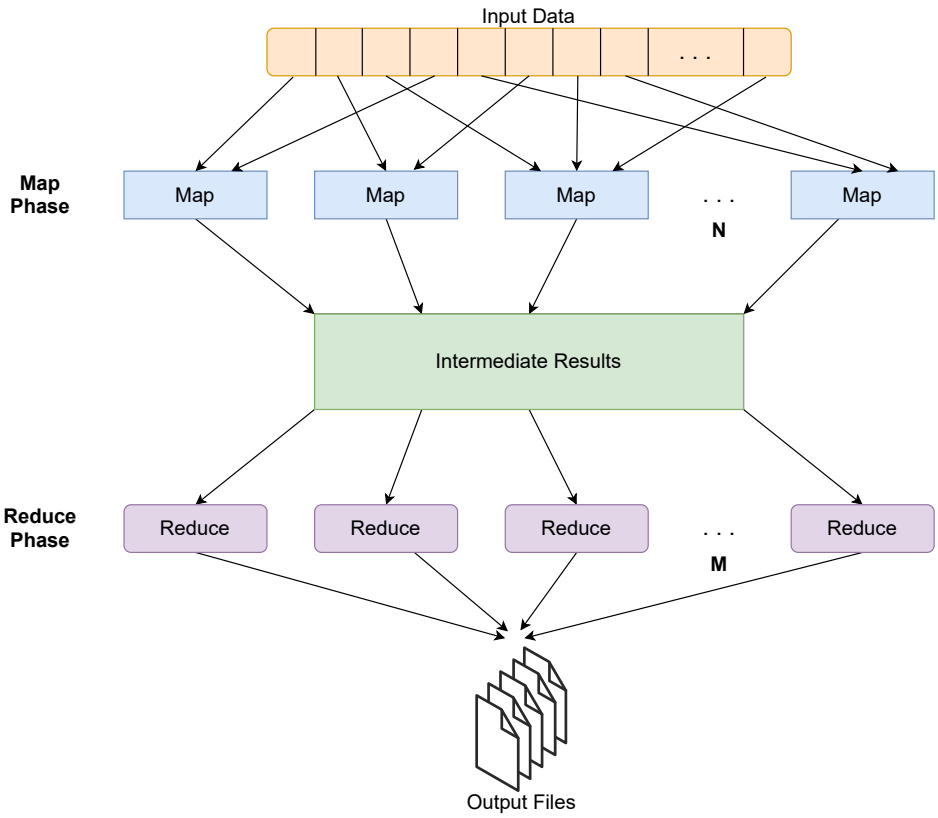
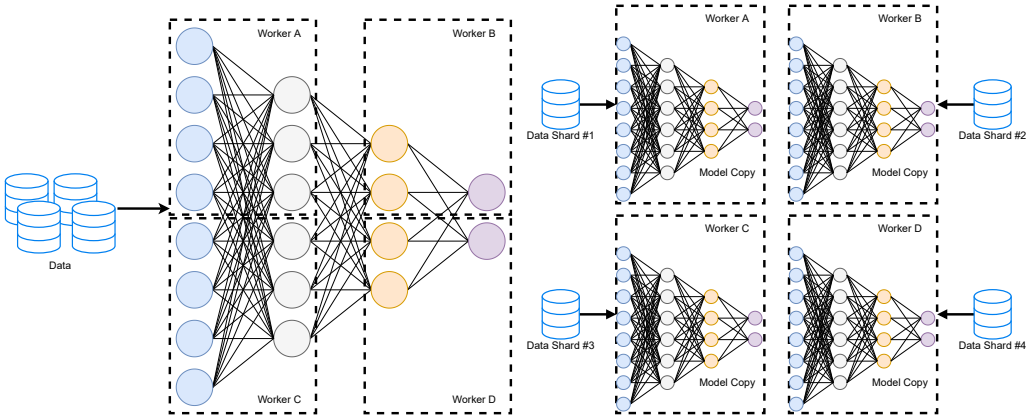


Figure 2.1: The Map-Reduce programming model.

The Map-Reduce programming model was first applied to machine learning problems in 2006 [72]. The authors demonstrated that, by adhering to the Statistical Query Model [73], many algorithms can be re-expressed in a form suitable for Map-Reduce (see Figure 2.1). Neural network backpropagation [74] and a range of simpler machine learning algorithms (*e.g.,* logistic regression [75], K-Means [76], etc.) are among those that fit the Statistical Query Model.

Since then, Map-Reduce has gained considerable attention in the machine learning community. For example, in 2009, researchers [77] proposed training conditional maximum entropy models at scale by using one Map-Reduce job per training iteration. However, concerns arose regarding the scalability of this approach and whether the resulting models matched the performance of single-node training. Motivated by these issues, Yahoo proposed a parallelization of Stochastic Gradient Descent [78] under the Map-Reduce framework in 2010, offering an alternative strategy for scaling machine learning tasks.



(a) Model Parallelism (b) Data Parallelism

Figure 2.2: Different types of distributed model training.

2.1.2 Specialized Architectures and the Parameter Server

While various MapReduce-based approaches have been explored in the learning domain, their iterative nature contrasts with the core principles of the MapReduce paradigm [79]. Consequently, specialized architectures more suitable for machine learning have emerged. Generally, learning-oriented distributed architectures rely on two types of parallelism: model parallelism and data parallelism.

Model parallelism [38, 39] applies when a model is too large to fit into a single machine's memory. In this setup, each worker in the training process holds only part of the model, as shown in Figure 2.2a. Conversely, data parallelism [40] is used to handle large datasets by splitting them into shards assigned to different machines. As illustrated in Figure 2.2b, each machine trains the same global model using only its assigned data. Many modern distributed deep learning systems support data parallelism, which can be applied regardless of the model [41].

Data-parallel training can follow one of two main architectures. One approach uses All-Reduce techniques [80] within a peer-to-peer network [81–83], as depicted in Figure 2.3. Another approach differentiates the participating workers into servers and workers according to the parameter server architecture. In this thesis, the focus is given on studying and optimizing the parameter server architecture, which is discussed in detail in the rest of this Section.

The Parameter Server [39, 57–59, 84, 85] is a widely used data-parallel architecture for training deep learning models in a distributed manner. This design targets machine learning algorithms that minimize a given optimization function, and is integrated into many state-of-the-art deep learning systems, such as TensorFlow [46, 86] and MXNet [47]—while others, such as PyTorch [64, 87], provide the necessary building blocks for engineers to implement it.

Within this architecture, each machine is designated either as a parameter server or a worker. Parameter servers store the model's parameters; one or more

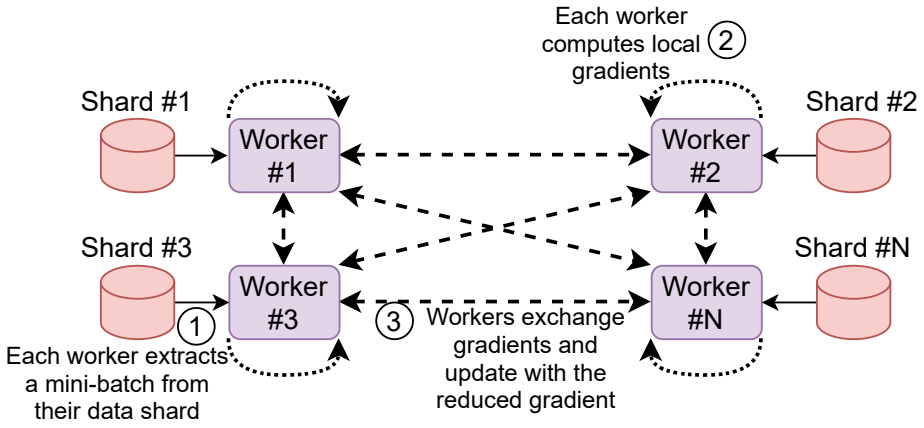


Figure 2.3: All-Reduce model training.

parameter servers may be used to mitigate communication bottlenecks on a single server node. Meanwhile, workers are tasked with gradient computation. The parameter servers update the global model by applying these gradients.

Figure 2.4 illustrates the training process with parameter servers and workers. The parameter servers divide (shard) the global model among themselves, with each server responsible for maintaining a subset of the parameters. The training dataset is similarly split among the workers, each of which keeps a local copy of the model and computes gradients using a mini-batch of its assigned data. These gradients are then sent (pushed) to the servers, which use the selected optimization algorithm to update the global model. After pushing their gradients, the workers pull the updated model to continue training. This procedure repeats until the model converges or reaches a user-specified number of iterations.

Hyperparameter Tuning

Choosing hyperparameters for deep learning models is a challenging process because the best values often depend on the characteristics of the dataset [88]. Key hyperparameters include those that affect the network architecture (e.g., number of layers) and those that influence training (e.g., mini-batch size B and learning rate α).

Even if optimal hyperparameters are identified for single-node training, these values may not be directly applicable to distributed training. However, when using the parameter server architecture, crucial hyperparameters, can be further derived to a per-worker level. For instance, this stands for the mini-batch size per worker (WB) and the learning rate per worker (α_w), which can be derived from their single node counterparts [89].

Suppose that the best hyperparameters for single-node training are a mini-batch size B and a learning rate α. Consider a cluster that employs the parameter server

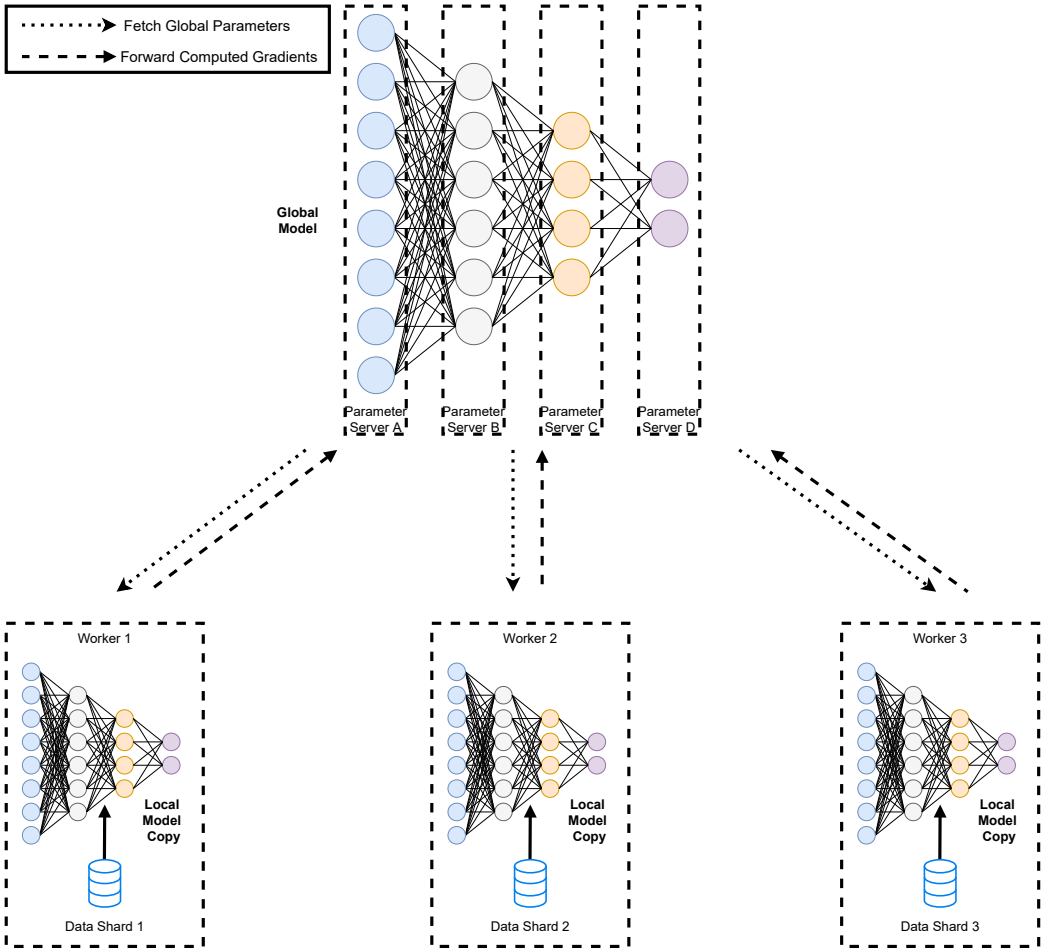


Figure 2.4: Parameter Server Architecture.

architecture with n workers. Since the product of the mini-batch size per worker WB and the number of workers must remain constant, the following equation can be derived:

$$B = n \cdot WB \quad \Longleftrightarrow \quad WB = \frac{B}{n}. \tag{2.1}$$

Similarly, the learning rate α_w for the distributed setup can be calculated from α by:

$$\alpha_w = \frac{\alpha}{n}. \tag{2.2}$$

Example 2.1 (Practical calculation) *Assume single-node tuning yields a mini-batch size* $B = 128$ *and a learning rate* $\alpha = 0.10$. *Deploying the same model on a cluster with* $n = 4$ *workers gives*

$$W_B = \frac{B}{n} = \frac{128}{4} = 32, \qquad \alpha_w = \frac{\alpha}{n} = \frac{0.10}{4} = 0.025.$$

*Each worker therefore processes **32 data points per step** while using a **quarter-sized learning rate**, preserving the effective 128-data batch and keeping gradient magnitudes comparable to the single-node run.*

Synchronization

In a parameter server setup, training can be performed either synchronously or asynchronously. Under the synchronous protocol, the system follows the Bulk Synchronous Parallel (BSP) model [90], where there is a barrier at the end of each iteration. This barrier ensures that parameter servers have received gradients from all workers before updating the global model. Consequently, at the beginning of the next iteration, each worker pulls a model copy that reflects all updates from the previous iteration.

Without synchronization constraints, the parameter server can also operate in an Asynchronous Parallel (ASP) manner. In this scenario, each worker computes gradients based on the most recently received global parameters. As a result, during the same training step, different workers may be using different versions of the global parameters. Formally, if $\vec{w}_\theta$ denotes the parameters retrieved from the server by a worker, and $\vec{w}_k$, $\vec{w}_{k+1}$ are the current and the updated global parameters, respectively, then equation A.2 is rewritten as:

$$\vec{w}_{k+1} = \vec{w}_k - \frac{\alpha_w}{WB} \cdot \sum_{i=1}^{WB} \nabla_{\vec{w}_\theta} L(\vec{w}_\theta; \vec{x}_i, y_i), \tag{2.3}$$

where α_w and WB are adapted as described in Section 2.1.2.

2.2 Architecture of Representative Systems

In this section, an overview of the architectures of TensorFlow and Spark MLlib is presented. Table 2.1 presents the basic features of the two systems that are discussed in more detail in the following paragraphs.

2.2.1 Google TensorFlow

TensorFlow [63] is an open-source, scalable machine learning platform developed by Google. TensorFlow is the successor to DistBelief [39], a distributed system for neural network training used by Google since 2011.

Abstract Programming Model

TensorFlow operates on top of dataflow graphs that represent both the computation in the machine learning algorithm as well as the state on which the algorithm operates. The edges of the graph carry data modeled as *tensors* (multi-dimensional arrays) between different nodes in the graph. The nodes in the graph represent

Table 2.1: TensorFlow vs. Spark MLlib

	TensorFlow	Spark MLlib
Abstract Programming Model	DAG -based	DAG -based
Execution Model	Long-running parameter server and worker tasks	Spark jobs with short-running map and reduce tasks
Training Modes	Synchronous / Asynchronous	Synchronous
Data Access	Can avoid fetching unnecessary data from disk/ Supports caching	Fetches the whole training set from disk/ Supports caching

units of local computation on the input tensors, called *operations*. Examples of operations are matrix multiplication and convolution among others.

Execution Model

TensorFlow is deployed as a set of *tasks* which are processes that can communicate over a network following the parameter server architecture, which is thoroughly described in section 2.1.2. The parameter server tasks maintain the current version of the globally shared model parameters. The worker tasks, on the other hand, are responsible for performing the bulk of the computation on top of their local training data.

For example, in the context of SGD with a minibatch of T training examples and N worker tasks, each worker task computes gradients based on a *local sub-batch* consisting of T/N local training examples, as explained in section 2.1.2, and then updates the shared parameters hosted by M parameter server tasks.The process is depicted in Figure 2.5. Steps 1-3 and 4-5 are executed in parallel on the worker and the parameter server tasks respectively. Each worker gets the latest model parameters from the parameter server (Step 1), performs the local gradient computation (Step 2) and then sends the updated gradient to the parameter server tasks (Step 3). Sequentially, parameter servers update the model parameters with the received gradients (Step 4) and evaluate convergence (Step 5).

Training Modes

TensorFlow supports both *asynchronous* and *synchronous training*. In asynchronous training, each worker task performs the local training computation independently without coordinating with the remaining worker tasks and updates the global shared model parameters without using any locking mechanism. In synchronous training, on the other hand, workers read the same values for the current

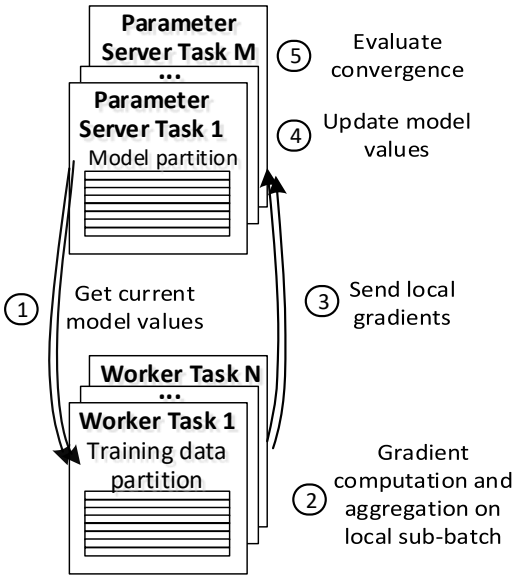


Figure 2.5: TensorFlow Execution Model.

model parameters until the parameter server receives the necessary user defined number of gradients to perform an update. At this point, the parameter server tasks aggregate all the computed gradients together.

Asynchronous Training is performed by allowing Step 4 in Figure 2.5 to be executed by the parameter server tasks whenever a worker task sends a new gradient update (Step 3). On the other hand, in the Synchronous mode, Step 4 is executed only when the parameter server has received one gradient update from each worker, which it aggregates into one using a *sum* operator. For this case study, TensorFlow was configures in an appropriate way to perform updates only after it receives one local update from each worker.

Data Access

In order to fetch data from disk, TensorFlow comes with an API, called Dataset API which provides a set of functions that can be sequentially used to create a pipeline that fetches, decodes and creates data batches, which are subsets of the data. These batches will be consumed from each worker to compute the next gradient updates.

The pipeline operates as follows: First, data is loaded from disk as raw bytes. Afterwards, the data rows are *mapped* to their decoded Tensor format in parallel. Finally, a *batch* is created. During these operations, *caching*, *batch prefetching* and

data shuffling is performed. These operations and their performance implications are discussed in the following sections. Among those three utilities, the first two crucially affect the performance, as will be described in Section 2.3.2, while the last one is used in SGD for random batch extraction.

2.2.2 Spark MLlib

Spark MLlib [91] is Spark's scalable machine learning library. Its basic features are described in this subsection.

Abstract Programming Model

Since MLlib is part of the Spark framework, it employs Spark's computation model. More specifically, Spark operates on DAGs whose nodes represent computations and edges represent *Resilient Distributed Datasets (RDDs)*. More specifically, Spark performs a series of computations following the Map Reduce model, described in Section 2.1.1, upon Resilient Distributed Datasets (RDDs). The RDD is a basic abstraction in Spark that represents an immutable distributed collection of data.

Execution Model

As opposed to TensorFlow that employs long-running processes, the machine learning algorithms executed through MLlib are deployed as a sequence of Spark jobs that consist of *map* and *reduce* stages whose corresponding tasks are launched by *Spark executors*. State is maintained across jobs through the *Spark driver* which is the main program of the Spark application. The Spark driver makes use of *broadcast variables* [92] to transfer state across the launched Spark jobs. The training data is typically represented as a partitioned RDD which is a collection of elements that can be operated on in parallel [93].

In the context of the SGD algorithm, a sequence of Spark jobs are launched, each one processing a minibatch, until the algorithm converges. The process is depicted in Figure 2.6, where dashed and solid arrows denote the control flow and data flow respectively. First, the Spark driver initializes a Spark broadcast variable that contains the current values of the model parameters (Step 1). In the next step, a Spark job is launched which is responsible for generating a minibatch by randomly selecting a subset of training examples from the partitioned RDD and processing the selected minibatch (Step 2). In particular, the Spark executors launch the map and reduce tasks across the nodes of the cluster. The map tasks compute the gradients on top of their local RDD partitions (Step 3). Note that during the gradient computation, the map tasks consider only the local training examples that belong to the selected minibatch as depicted by the black lines in Figure 2.6. After all the local gradients have been computed, the reduce tasks perform an aggregation of the partial gradients (Step 4). The Spark driver collects

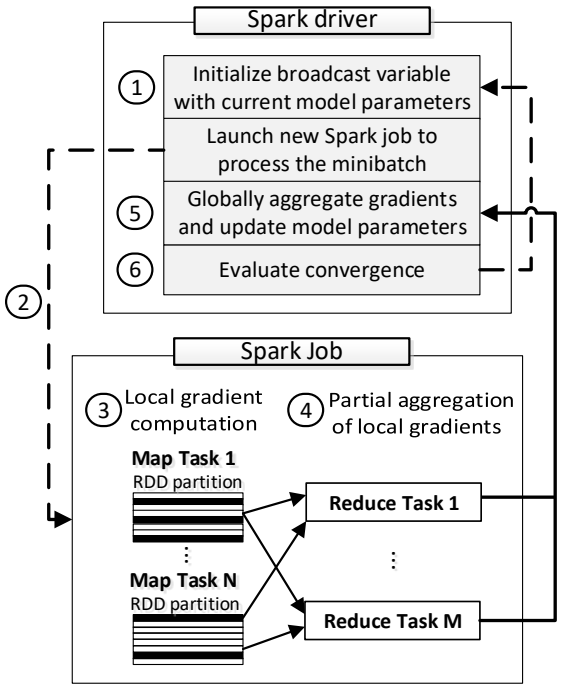


Figure 2.6: Spark MLlib Execution Model.

the output of the reduce tasks, performs the global aggregation and maintains the updated model parameters (Step 5). Finally, the Spark driver evaluates the convergence criteria and repeats the above steps if needed (Step 6).

Training Modes

MLlib supports only synchronous training in contrast with TensorFlow, because of its MapReduce-based execution model.In particular, the output of all the map tasks is required before the reduce tasks perform any aggregation and before the Spark driver updates the model parameters.

Data Access

Since Spark employs lazy evaluation [93], the partitioned RDD that points to the training data will be fetched from disk only when the map tasks perform their local computations. In particular, every time a new Spark job is launched to process a minibatch, the map tasks fetch their local RDD partitions from disk while performing the gradient computations. Note that as opposed to TensorFlow where the built-in input pipeline fetches only the local sub-batches accessed by the

worker task, Spark requires the RDD partition to be fully read even if the map task will eventually perform computations based on a subset of the training examples (minibatch). To avoid the overhead of repeatedly fetching the data corresponding to each RDD partition from disk every time a new Spark job is launched, the local RDD partitions are cached in the memory of each node. As a result, the subsequent map tasks launched at the node will read the data from local memory.

2.3 Experimental Evaluation between Map-Reduce and Parameter Server

2.3.1 Experimental Setup

Hardware and Software Configuration

The experiments are performed on a cluster of 141 virtual machines ("nodes") in Okeanos public cloud [94, 95]. Each virtual machine has 2 virtual CPUs @ 2.1GHz, 8 GB of RAM and 20 GB of hard disk storage. One of these nodes is used as *master node* and the remaining ones as *worker nodes*. Thus, the worker nodes provide a total of 280 virtual CPUs, 1 TB of RAM and 2.8 TB of hard disk storage.

The operating system used is Debian Jessie 8.10. TensorFlow version 1.13 and Spark version 2.4 are used , which were the latest available versions at the time of running the benchmarks. In the case of TensorFlow, one worker task is deployed on each of the 140 worker nodes. One parameter server task is deployed at the master node. In the case of MLlib, one Spark executor is deployed on each of the 140 worker nodes. The Spark driver is located at the master node.

Machine Learning Models

Three models for predictive analysis, namely *linear regression* [96], *binary logistic regression* [97], and the *multilayer perceptron (MLPC)* classifier [98] are being used.

The *linear regression* model is frequently used for regression problems where the goal is to model the relationship between a scalar dependent variable and a set of independent variables. The *binary logistic regression* model is a regression model with a categorical dependent variable which is used to solve classification problems. Finally, the *perceptron* classifier is a binary classifier based on a feedforward artificial neural network that consists of multiple layers of nodes.

The *perceptron* classifier is chosen, since it is a representative from the deep learning subdomain and is implemented in both systems. Specifically, the corresponding artificial neural network consists of 4 layers, two of which are the hidden ones, with 28, 15, 15 and 2 neurons respectively. GD and SGD optimizers were used in the training, since they are widely used and supported by both systems.

Table 2.2: Real datasets

Dataset	# Examples	# Features	Size
HIGGS	10,500,000	28	2.7 GB
Year_Prediction_MSD	470,000	90	0.391 GB

However, for perceptron, only GD is used, since SGD is not supported by MLlib for this algorithm.

Note that as opposed to MLlib which provides an out-of-the-box implementation of the above models, TensorFlow also provides the building blocks for writing machine learning algorithms. While it provides implementations of the two regression algorithms, they differ from the MLlib ones and also do not support convergence check. To perform a fair comparison between the two systems, these building blocks were utilized to implement all the models in TensorFlow in the same way that they are implemented in MLlib. To compute the model parameters, an optimization algorithm is used to minimize a loss function. The GD and SGD optimization algorithms are used since they are widely used and they are supported by both systems. For the *linear* and *logistic* regression models, experiments with both GD and SGD are conducted, while for the *perceptron* classifier only the GD algorithm is used as SGD is not supported in MLlib for this model.

Methodology

The provided experimental evaluation consists of two parts. First, synthetic data are generated, on which a series of experiments is performed in a 141-node cluster, in order to examine how both systems operate at large scale. Since evaluating convergence on top of synthetic data is not recommended, the number of minibatches that the algorithms process before terminating is fixed. Furthermore, experiments using real datasets in a 5-node clusterare conducted in order to study convergence. Note that a small cluster is used, since real datasets are not large enough to fully utilize a larger one. For TensorFlow, both the synchronous and the asynchronous training modes are examined in each experimental section.

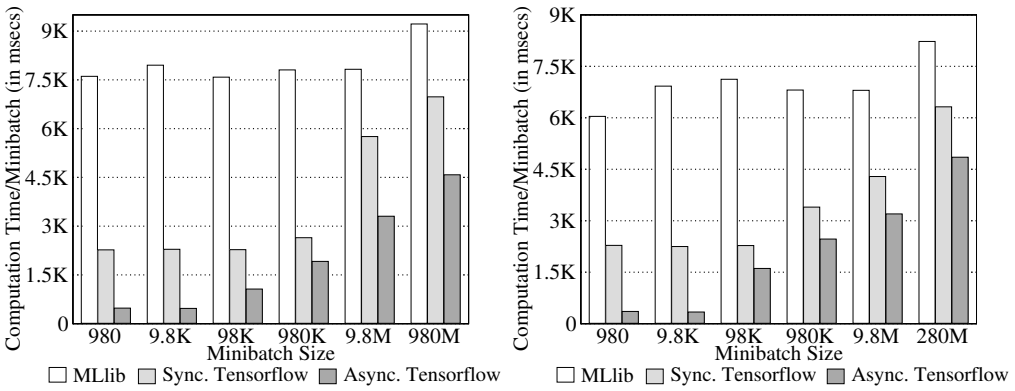
Datasets

For the experiments with real data, datasets from the UCI Machine Learning Repository [99] are used, which are stored as comma separated text files. More specifically, the `HIGGS` [100] dataset for the *logistic regression* and *perceptron* models and the `Year_Prediction_MSD` [101] dataset for the *linear regression* model are used, which were the largest real ones without missing values in the repository. Table 2.2 presents the characteristics of these datasets.

To generate the synthetic data, each real dataset is replicated multiple times so that the resulting dataset barely fits in the memory of the cluster (see Table 2.3).

Table 2.3: Synthetic datasets

ML Model	# Examples	# Features	Size
Logistic Regression	980,000,000	28	252 GB
Linear Regression	280,000,000	90	238 GB
Perceptron	441,000,000	28	110 GB



(a) Logistic Regression (b) Linear Regression

Figure 2.7: Computation time per minibatch in MLlib and TensorFlow (synchronous and asynchronous) on a 141-node cluster.

This is because, for the GD algorithm, TensorFlow requires the whole dataset to fit in the aggregate memory of the cluster in contrast with Spark.

The datasets are stored in HDFS when MLlib is used and are split into equal parts that are uniformly distributed across all the cluster nodes when TensorFlow is used.

2.3.2 Experiments on Large Synthetic Datasets

In this section, experiments on the 141 node cluster using synthetic data are presented (Table 2.3), to identify the performance bottlenecks of each system by carefully profiling them on a large-scale. As described in Section 2.3.1, the training process is terminated after the algorithms process a fixed number of minibatches, which are set to 100 for the *logistic* and *linear regression* models and to 20 for the *perceptron* model.

Logistic and Linear Regression

Figure 2.7 shows the average time spent by each system in performing gradient computations when processing a minibatch using *logistic* and *linear regression*.

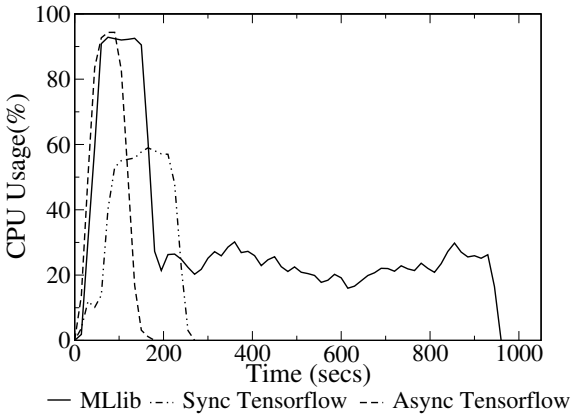


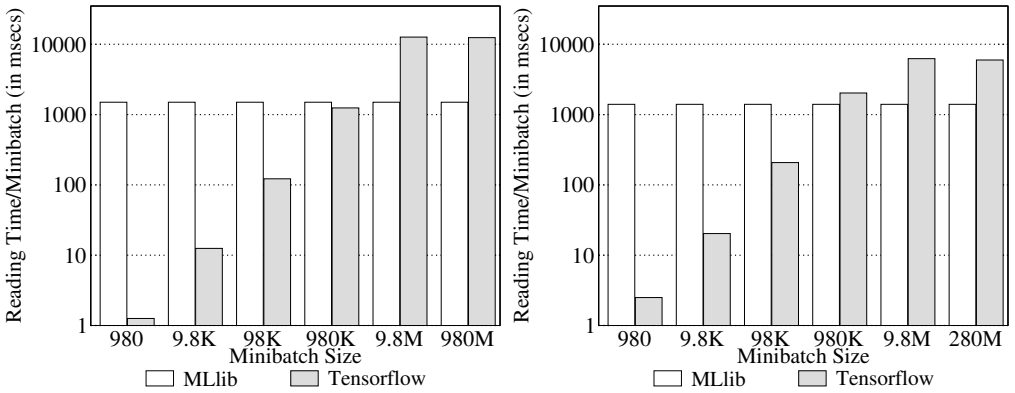
Figure 2.8: Cluster CPU usage for Logistic Regression (minibatch=980K)

Three bars are presented for each minibatch size, referring to Spark MLlib, synchronous and asynchronous Tensorflow.

As shown in Figure 2.7, TensorFlow spends always less time for computing gradients regardless of the training mode. As minibatch size decreases, TensorFlow is more computationally efficient than Spark MLlib. Moreover, when decreasing the minibatch size, TensorFlow achieves up to $14X$ speedup, whereas Spark becomes faster up to $1.36X$. Compared to TensorFlow's long-running processes, Spark launches a new Spark job every time a new minibatch is processed. This introduces task scheduling and initialization overheads (especially when the minibatch decreases and map tasks become much shorter ($\approx$100msecs)), and low CPU utilization, as confirmed in Figure 2.8. The same figure suggests that synchronous TensorFlow also suffers from some overheads but these are mostly synchronization overheads. Despite these overheads, it still has better CPU utilization than Spark.

Figure 2.9 show the average time spent by each system in reading data (fetch, deserialize and decode raw bytes) when processing a minibatch using *logistic* and *linear regression*. Reading time is presented for MLlib and TensorFlow irrespective of its training modes, since it does not depend on them.

Regarding the average reading time per minibatch, Spark MLlib spends the same amount of time reading data irrespective of the minibatch size, as it reads the whole RDD partition in all cases. TensorFlow on the other hand, fetches only the data needed for the current gradient computation. As a result, in TensorFlow the average reading time drops when the minibatch sizes decreases. However, in the *linear regression case*, Tensorflow spends almost the same time reading each minibatch for both GD and SGD (minibatch size 9.8M) as shown in Figure 2.9b. Since each algorithm runs until the system processes 100 minibatches, the full dataset is read by both algorithms (see dataset size in Table 2.3). In SGD, each row is cached after it has been decoded and before it is used for minibatch creation. On the contrary, in the case of GD, it is optimal to cache the whole minibatch instead



(a) Logistic Regression (b) Linear Regression

Figure 2.9: Reading time per minibatch in MLlib and TensorFlow on a 141-node cluster

Table 2.4: Read Throughput of Spark MLlib and TensorFlow

ML Model	**Read Throughput (training examples/sec)**	
	Spark MLlib	**TensorFlow**
Logistic Regression	6,533,333	785,323
Linear Regression	2,000,000	456,774
Perceptron	1,986,486	788,048

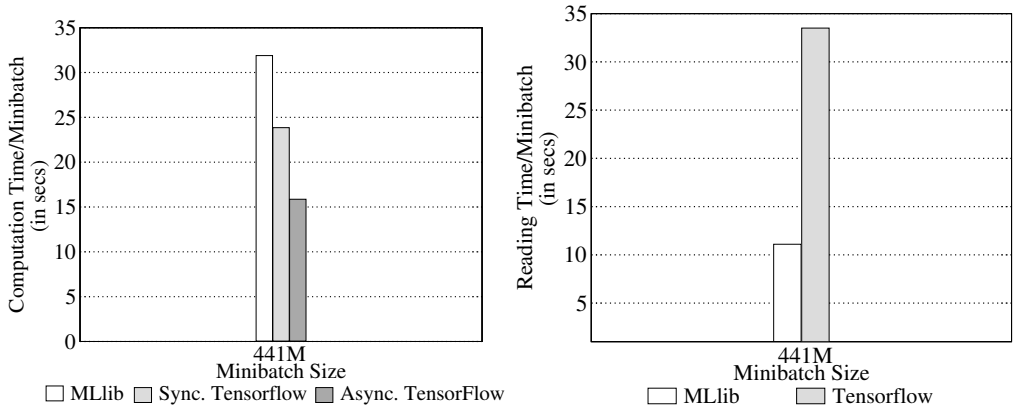
of a row-at-a-time, resulting in $1.62X$ speedup in reading time and achieving a similar performance as SGD.

Perceptron

Figure 2.10 presents the computation and reading time spent per minibatch for *perceptron*. Since the full dataset is used as a minibatch (GD), TensorFlow spends more time reading data than Spark. As shown in Figure 2.11, TensorFlow, especially in synchronous mode, suffers from reading the whole dataset before the first gradient computation: unlike asynchronous mode, the system is blocked until all their workers finish reading their local data depicted by the CPU usage drop between 500 and 700 sec.

Read Throughput

To better understand the reading performance, the read throughput of each system is computed and presented in Table 2.4. As shown in the table, the throughput varies across different algorithms depending on the number of features and the



(a) Comp. Time per Minibatch (b) Reading Time per Minibatch

Figure 2.10: Times per minibatch spent for Perceptron in MLlib and TensorFlow on a 141-node cluster

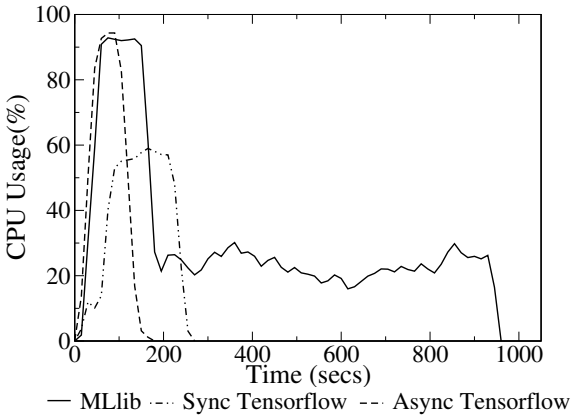
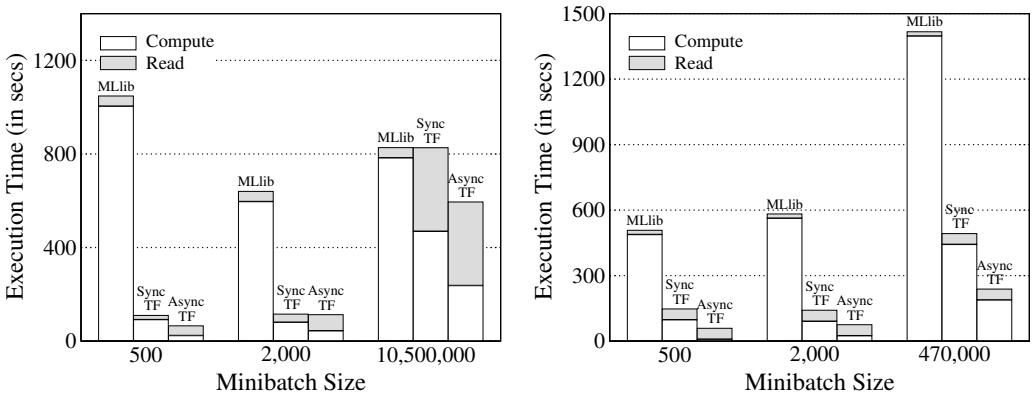


Figure 2.11: Cluster CPU usage for Perceptron

transformations required for each dataset. However, Spark always reads data faster and has up to $8X$ better reading throughput than TensorFlow.

2.3.3 Experiments on Real Datasets

In this section, experiments using the real datasets are presented in Table 2.2 on a 5-node cluster. As noted before, experiments on a larger cluser could not be run as the real datasets are generally small. All the algorithms are executed until they have converged.



(a) Logistic Regression (b) Linear Regression

Figure 2.12: Logistic and linear regression performance of Spark MLlib, synchronous (Sync TF) and asynchronous TensorFlow (Async TF) on a 5-node cluster

Table 2.5: Number of minibatches processed per system for logistic and linear regression to converge

ML Model	**Minibatch Size**	**# Minibatches Processed**		
		MLlib	**Sync TF**	**Async TF**
Logistic Regression	10,500,000	317	317	137
	2,000	774	737	1006
	500	1375	1644	2449
Linear Regression	470,000	3104	3104	1328
	2,000	2678	2773	1042
	500	2438	2540	1121

Linear and Logistic Regression

Figures 2.12a and 2.12b present the total execution time of TensorFlow and Spark MLlib for the *logistic regression* and *linear regression* algorithms respectively, including the portion of the time spent in reading data and performing computations in each system. The various minibatch sizes are selected to be realistic [102].

As shown in the figures, TensorFlow is faster than Spark MLlib by up to $16X$ when the SGD algorithm is used (Figure 2.12a, minibatch size 500). This behaviour is mainly attributed to architectural differences of the two systems, as explained in Section 2.3.2. Regardless of the training mode, TensorFlow presents at worst the same performance as MLlib, with the asynchronous mode being faster.

Table 2.5 shows the number of minibatches that each system processes before the algorithm converges.While the number of minibatches is the same between

synchronous TensorFlow and MLlib in the case of GD as expected, a small deviation is observed in the SGD case. This is due to the fact that each system performs batch selection differently as explained in Section 2.2, resulting to dissimilar data per minibatch.

Another interesting observation is that the number of minibatches processed by GD and SGD is different for the same machine learning model, as SGD might need to process more minibatches until convergence [102] (see *logistic regression*).

As explained in section 2.3.2, TensorFlow has different behaviour in the two training modes when the minibatch size decreases. For instance, as shown in Figure 2.12a in the context of *logistic regression*, asynchronous training is $1.72X$ faster when reducing the minibatch size from 2000 to 500, since the system processes $1.65X$ fewer rows in total. On the contrary, synchronous training needs almost the same time to converge with the two batch sizes. Similar trends are noticed in the case of Spark MLlib. For example, in the case of *linear regression*, Spark needs almost the same time to process $4.02X$ more rows until convergence, when increasing the minibatch size from 500 to 2000. This behavior is attributed to reading the whole dataset at every gradient computation irrespective of the minibatch size and to various other computational overheads as discussed in Section 2.3.2.

However, Figures 2.12a and 2.12b show that SGD with a minibatch size of 2000 is $1.29X$ and $2.43X$ faster than when GD is used. Thus, Spark meets its best performance when the batch size is such that the overheads mentioned above do not dominate the execution time.

Perceptron

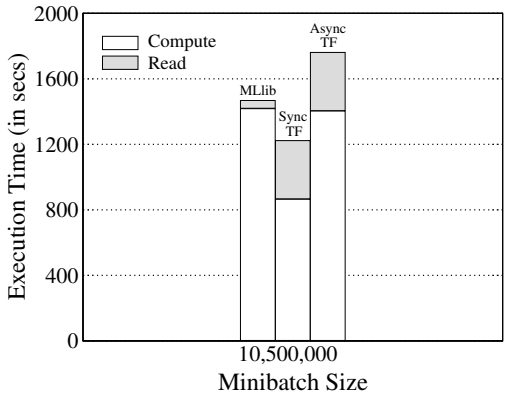


Figure 2.13: Perceptron performance of Spark MLlib, synchronous (Sync TF) and asynchronous TensorFlow (Async TF) on a 5-node cluster.

Figure 2.13 presents the total execution time of the systems on the *perceptron* classifier. Note that only the GD optimization algorithm is used as discussed in Section 2.3.1.

Table 2.6: Number of minibatches processed per system for perceptron to converge

ML Model	Minibatch Size	MLlib	Sync TF	Async TF
Perceptron	10,500,000	72	72	118

As Figure 2.13 outlines, Spark MLlib is $1.2X$ faster than asynchronous TensorFlow for this algorithm. However, by taking a closer look at the number of minibatches processed in Table 2.5, TensorFlow appears to have processed almost twice the number of minibatches that Spark MLlib processed before converging. This behavior is due to the asynchronous training of TensorFlow: since the worker tasks might overwrite each other's work, more computation steps are required before the algorithm can converge.

In synchronous mode, TensorFlow needed $1.2X$ less time to converge with the same number of minibatches with MLlib, while its reading phase is $7.2X$ slower, which confirms its computational efficiency.

2.4 Key findings and conclusions

Overall, the general-purpose engine Spark is superior during data reading (loading raw bytes, decoding and deserializing). The specialized engine (TensorFlow) outperforms Spark's MLlib during gradient computation, mainly due to the fundamental architectural differences of the two systems. in further detail, the key findings of this experimental evaluation are the following:

1. TensorFlow is more computationally efficient than Spark MLlib regardless of the training mode (see Figures 2.7 and 2.10a), but Spark has better read throughput (see Table 2.4).

2. Spark's short-running tasks, that perform computations and model updates, introduce scheduling and initialization overheads, especially for small minibatches (see Section 2.3.2). TensorFlow's long-running processes avoid such overheads under the parameter server approach.

3. TensorFlow needs less time to converge than Spark MLlib in almost all the machine learning models that were examined (see Figure 2.12.

4. Spark meets its best performance when the minibatch size is such that the initialization overheads of Spark are amortized. (see Figures 2.12a and 2.12b).

5. Spark reads more data than TensorFlow when the SGD algorithm is used which further hurts its performance (see Figure 2.9)

Chapter 3

Diving into the Parameter Server: A survey on recent advances

The experimental evaluation of the *Parameter Server* architecture in Chapter 2, proved its computational efficiency in training workflows, in contrast with using general-purpose architectures. The rest of this thesis will focus in researching and optimizing the *Parameter Server* architecture.

This chapter provides an extensive survey of the parameter server architecture, including a historical overview and an examination of existing approaches that address its various challenges. By analyzing the vulnerabilities of the architecture and how they arise, the groundwork for Chapters 4-6 is laid, where optimizations for the *Parameter Server* architecture are introduced.

Several existing surveys have addressed specific aspects of distributed training systems and parameter server architectures. For example, Ratnaparkhi et al. [103] and Zhang et al. [104] provide an overview of distributed learning frameworks, but their treatment of parameter servers is limited to brief mentions. Later works, such as Verbraeken et al. [105], focus on synchronization techniques (e.g., BSP, ASP, and SSP) and their trade-offs, while Shi et al. [106] examine communication optimizations in parameter servers. Other studies delve into specialized topics, such as gradient compression [107]. Despite these contributions, there remains a gap in providing a holistic review of parameter server architecture. Specifically, existing surveys tend to focus on narrow subfields, lacking a comprehensive examination of its key aspects, such as:

- **Consistency Control**: Methods for balancing synchronization and efficiency.
- **Network Optimization**: Techniques to mitigate communication bottlenecks.

- **Parameter Management**: Strategies for efficient storage and load balancing.
- **Straggler Problem**: Solutions to address task delays in heterogeneous environments.
- **Fault Tolerance**: Approaches for maintaining reliability under node failures.

In this survey, these gaps are addressed by systematically reviewing the literature on parameter server architectures. It highlights key advancements in the field, identifies open challenges, and provides future research directions. By offering a unified perspective, this survey positions itself as a comprehensive resource for researchers and practitioners interested in distributed learning. More specifically, research topics and advances related to the parameter server architecture are described and presented as the primary focus. The main contributions of this work are the following:

- After a thorough bibliographical study following the SLR methodology [108], the research works that study and optimize the parameter server approach are categorized into five large general topics.
- An extensive literature review is provided for each of the identified parameter server research topics.

The rest of this chapter is organized as follows: Section 3.1 presents some historical information regarding the evolution of parameter servers. Section 3.2 presents the surveyed literature grouped by the improved aspect into categories.

3.1 Parameter server history

Before parameter server was proposed as an architecture, Smola introduced a general architecture for parallel models in 2010 [57], where key-value storage was used to maintain some global state of a problem. A variety of workers synchronized periodically to keep up to date the global state based on their computations. A year later, *Hogwild!* [109] was proposed as an approach to run SGD in parallel without synchronization in multicore environments. However, *Hogwild!* faced gradient collisions, where the movement of the network parameters attributed to a set of gradients was overwritten by another set.

Considering Smola's parallel architecture and *Hogwild!* as its ancestors, parameter server as a specialized distributed learning architecture finds its roots back in 2012 when Google Research proposed *DistBelief* [39], as a "*software framework that can utilize computing clusters with thousands of machines to train large models*". In the context of *DistBelief*, the authors proposed an alternative to the

Table 3.1: Well-known systems adopting the parameter server architecture

System	Developed By	Since	Language	APIs	Parameter Server Development Level	Sync. Level
Petuum	Carnegie Mellon University	2015	C++	C++, Java	Implemented	(Bounded) Asynchronous
PyTorch	Meta AI	2016	C++, Python	C++, Python	Building Blocks	Synchronous, Asynchronous
TensorFlow	Google Brain Team	2015	C++, Python	Python, Go, Java, C++	Experimental	Asynchronous (up to latest), Synchronous (v1 only)
MXNet	Carlos Guestrin (University of Washington)	2015	C++, Python, Perl	C++, Python, Scala, Java, etc.	Implemented	Synchronous, Asynchronous
DeepSpeed	Microsoft	2020	Python	Python	Complementary to PS Architectures	Primarily Asynchronous
Ray	Anyscale, founded by Berkeley's RISELab	2019	Python, Java	Python, Java	Hybrid with PS Functionality	Synchronous, Asynchronous
Horovod	Uber	2017	C++, Python, Go	C++, Python, TensorFlow APIs	Compatible with PS Architectures	Synchronous (via Ring All-Reduce)

classical SGD algorithm, namely Downpour SGD, which exploits the parameter server architecture, but in an asynchronous manner, as described in Section 2.1.2. As discussed, workers replicate locally the global model copy by fetching the latest model parameters, where they compute a set of gradients based on a local data shard. Afterwards, the workers push the gradient updates back to the servers over the network to update the global model. Asynchrony, as a state, is mentioned to result in computing gradients based on slightly outdated, or stale, model parameters, leading to the later known stale gradients effect, but is much faster and more fault tolerant in comparison with synchronous distributed SGD setups. Both the observations from the *DistBelief* and *Hogwild!* authors indicate that, while lacking synchronization can provide faster results, it may crucially affect the model quality opening the way to discuss various possible optimizations.

Over the last decade, parameter server is being widely adopted in many deep learning related systems. Table 3.1 highlights a range of real-world systems that implement the parameter server architecture, showcasing its adoption across academia and industry. These systems underline the flexibility and scalability of the parameter server paradigm, particularly in addressing the needs of large-scale distributed training.

TensorFlow [63, 86] and PyTorch [64, 87], developed by Google and Meta, respectively, incorporate parameter server functionalities to facilitate distributed training. TensorFlow includes experimental implementations of parameter servers, supporting both asynchronous and synchronous modes in earlier versions, whereas PyTorch provides building blocks for customization, enabling synchronous and asynchronous execution. These frameworks are widely used in industries for tasks like image recognition and natural language processing, demonstrating their broad applicability. MXNet [47], supported by AWS, offers robust parameter server implementations, focusing on scalability and performance in cloud-based machine learning pipelines. Similarly, Petuum [85], developed at Carnegie Mellon University, emphasizes efficient and scalable machine learning applications, leveraging bounded asynchronous synchronization for optimized training performance.

Recent frameworks such as Microsoft's DeepSpeed [44, 110] and Ray [53] have further enhanced distributed learning, integrating complementary functionalities for parameter server setups. DeepSpeed optimizes large-scale distributed training with features like ZeRO (Zero Redundancy Optimizer) [110, 111], while Ray provides a hybrid framework that supports parameter server paradigms through its task orchestration capabilities. Additionally, Horovod [112], a TensorFlow extension developed by Uber, introduces distributed gradient aggregation using *Ring All-Reduce*, which can complement parameter server architectures by decentralizing gradient synchronization.

The table encapsulates the progression and versatility of parameter server systems, demonstrating their integral role in modern distributed learning frameworks. From academic prototypes like Petuum to production-grade platforms like TensorFlow, these implementations highlight the practical effectiveness of parameter server architectures in diverse domains. This description integrates real-world performance and use cases, reflecting the architecture's industrial relevance.

3.2 Proposed Approaches

In the last decade, a wide variety of research works have attempted to enhance the parameter server architecture in various ways. This section presents works of high interest grouped by the aspect of the parameter server they aim to improve.

3.2.1 Systematic Literature Review on the Parameter Server

Parameter server, as proposed by *DistBelief*, is a very prominent approach to training large models with big data for a variety of reasons:

1. This architecture can support enormous models since the corresponding parameters can be split between the various deployed servers.

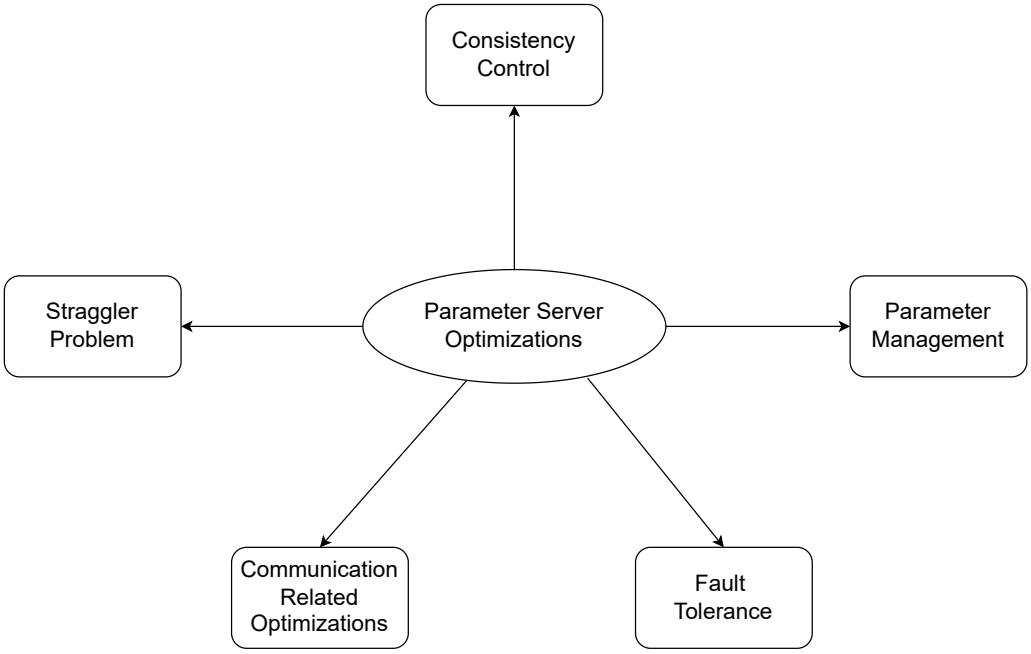


Figure 3.1: Parameter server research areas

2. Data parallelism can speed up the whole process of completing a training epoch.

For all the aforementioned reasons, parameter server is of high interest to researchers studying distributed systems and are interested in the learning domain. Depending on the field of interest, researchers can dive into various aspects that emerge from using this architecture. Figure 3.1 outlines the main research topics for optimizing the performance of the architecture, in which the rest of the current Section dives into.

To ensure a comprehensive and methodical review of this field, a systematic literature review (SLR) [108] is conducted, which involves the following steps:
1. Defining research questions (RQs): The SLR is designed to answer key research questions that align with the core themes of parameter server advancements:

- **RQ1:** What are the major synchronization strategies proposed for consistency control in parameter servers?
- **RQ2:** How have communication optimizations evolved to enhance scalability and efficiency?
- **RQ3:** What are the primary approaches for addressing the straggler problem and improving fault tolerance?

2. Literature search strategy: Research databases (e.g., IEEE Xplore, ACM Digital Library, Scopus) are systematically queried using the following search terms: "parameter server," "distributed machine learning," "synchronization protocols," "gradient compression," and "fault tolerance." The search covered peer-reviewed articles published between 2010 and 2024, as this period represents the emergence and maturation of parameter server architectures.
3. Inclusion and exclusion criteria Clear criteria to select relevant studies are established:

- **Inclusion criteria:** Peer-reviewed articles, studies focusing on parameter server architecture or its applications, contributions to consistency control, communication efficiency, or scalability.
- **Exclusion criteria:** Articles focusing solely on federated learning or unrelated distributed training architectures, non-English publications or inaccessible full-text articles.

4. Data extraction and categorization: The following data points are extracted from each selected study:

- Title, year, and venue.
- Proposed methodologies and experimental results.
- Key contributions to the parameter server field.

These findings were categorized into thematic areas, as outlined in the subsections below.

3.2.2 Consistency Control

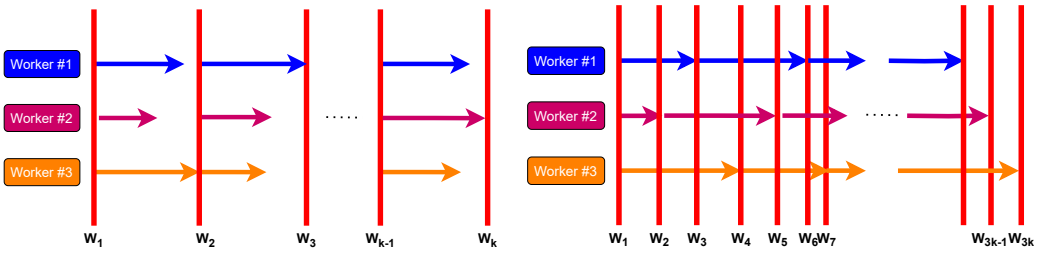
One of the most important topics regarding the parameter server is to identify how strict control needs to be applied in the model consistency. Basic approaches, as discussed back in Section 2.1.2, include the parameter server training working at the extremities of model consistency:

- A fully consistent model via synchronizing all participating workers at the end of each iteration using a BSP approach.
- A possibly inconsistent model with totally asynchronous workers per iteration utilizing an ASP approach.

In the opposite direction of BSP, ASP removes any synchronization constraint. As indicated in Figure 3.2b, whenever a worker completes an iteration, they push the computed gradients to the servers. Sequentially, the servers immediately update the global parameters, which are then pulled by the worker that pushed the

Table 3.2: Consistency Control Algorithms

Algorithm	Description
BSP (Bulk Synchronous Parallel) [90]	All workers synchronize at the end of every iteration, ensuring a fully consistent model. Delays may occur due to stragglers. (Figure 3.2a)
ASP (Asynchronous Parallel) [109]	Workers push gradients and pull updated parameters asynchronously, improving speed but risking model inconsistency. (Figure 3.2b)
k-BSP [89]	The server waits to collect gradients from k workers before updating the global model. Stalled workers' gradients are ignored. (Figure 3.3a)
k-Batch-BSP [89]	Similar to k-BSP, but the server waits for k gradient batches, potentially from the same worker, improving flexibility. (Figure 3.3b)
k-ASP [89]	The server collects k gradients asynchronously without canceling stalled workers, balancing speed and gradient utilization. (Figure 3.3c)
k-Batch-ASP [113]	Extends k-Batch-BSP by allowing asynchronous updates and retaining gradients from slower workers. (Figure 3.3d)
SSP (Stale Synchronous Parallel) [45]	Introduces bounded staleness where workers can lag behind the fastest worker by a predefined threshold, balancing consistency and speed. (Figure 3.4)

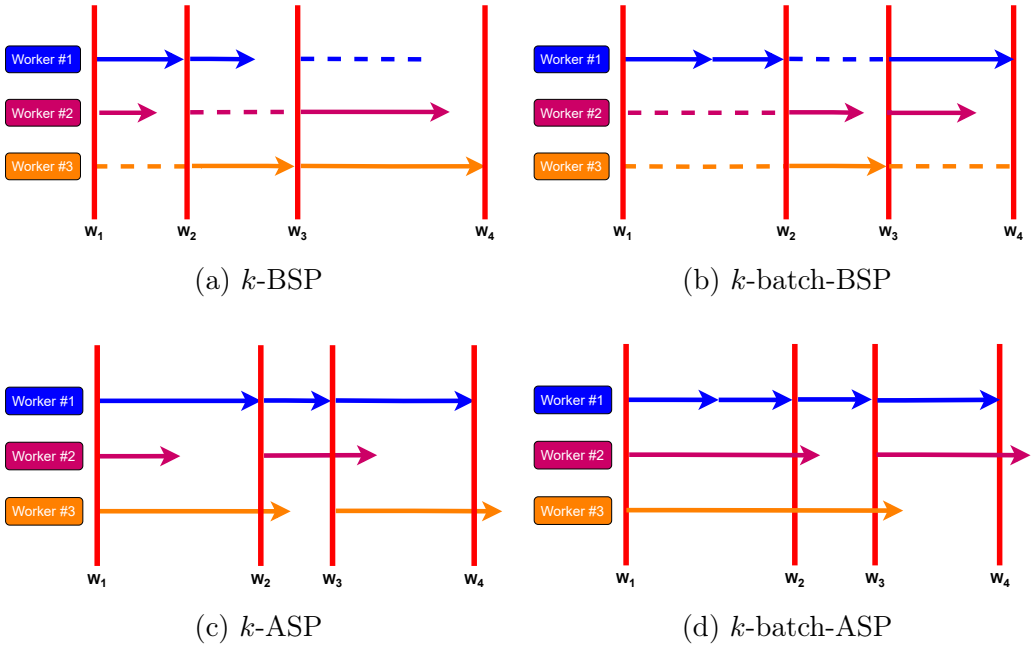


(a) BSP (b) ASP

Figure 3.2: Sequence diagram presenting 3 workers training under a BSP (left) and ASP (right) Parameter Server for k global iterations

gradient. In general, in this setup, $3k$ different model parameters will be computed during training, as k global iterations are needed. Although this setup is free of any overhead related to synchronization, workers can compute gradients from stale parameter values. An example could be provided using worker #2. When pulling w_2 from the servers, worker #2 stalls and pushes the computed gradients back after the server(s) are located in version 4 of the parameters. Therefore, the model might be inconsistent and less accurate than in the BSP case.

In the past decade, multiple systems have chosen to deploy the parameter server using either one of the BSP [114–121] and ASP [58, 59, 84] schemes or providing both for the user to choose [122–127]. While baseline techniques, these two approaches have inspired researchers to propose some alternatives that find similarities either to BSP or the ASP. Four variants directly inspired by these two synchronization schemes are the k-BSP, k-batch-BSP [89], k-ASP [89] and k-batch-ASP [113]. Figure 3.3a, Figure 3.3b, Figure 3.3c and Figure 3.3d outline the process of training under k-BSP, k-batch-BSP, k-ASP, k-batch-ASP for 3 workers



(a) k-BSP (b) k-batch-BSP

(c) k-ASP (d) k-batch-ASP

Figure 3.3: Sequence diagram presenting 3 workers training under k-batch-BSP (upper left), k-batch-BSP (upper right), k-ASP (down left) and k-batch-ASP (down right) for $k = 2$.

with $k = 2$.

- **k-BSP**: Figure 3.3a presents this case. Parameter servers need to collect gradients from $k = 2$ different workers before proceeding with updating the global model. Stalled workers in an iteration (*e.g.*, worker #3 after w_1 and worker #2 after w_2) are canceled from the parameter servers and their gradients are lost for this iteration.

- **k-batch-BSP**: Figure 3.3b outlines this approach, which is almost the same with k-BSP, with the difference that the servers wait to get $k = 2$ different batch gradient updates, which might not be necessarily from different workers. For example, after w_1, two update batches from worker #1 complete the first iteration, and both workers #2 and #3 are canceled for this step. On the contrary, the next iteration is fulfilled with the contributions of the two workers canceled before.

- **k-ASP**: Same as k-BSP. However, stalled workers are not canceled and their update might be one of the k gradients completing some of the next iterations, as shown in Figure 3.3c.

- **k-batch-ASP**. Equivalent to k-batch-BSP with the difference of not canceling the stalling worker (Figure 3.3d).

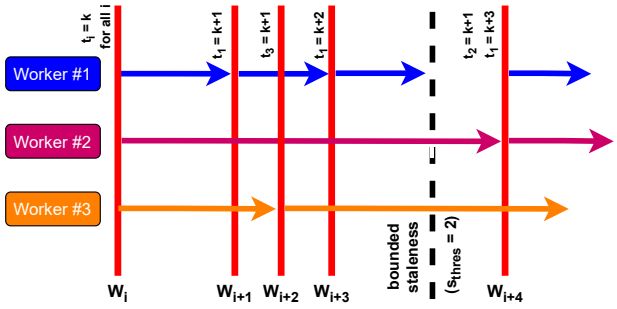


Figure 3.4: Sequence diagram presenting 3 workers training under different SSP for $s_{thres} = 2$.

One of the most prominent synchronization approaches to handle model consistency comes back in 2013, when Ho proposed the alternative of a *stale synchronous parallel* (SSP) approach [45]. This research work introduces the concept of *bounded staleness*. Followed by these principles, given each worker preserves a clock t_i indicating the moment j and a staleness threshold s_{thres}:

1. The clocks of the fastest ($t_{fastest}$) and the slowest ($t_{slowest}$) worker must differ at most by s, *i.e.*, $t_{fastest} - t_{slowest} \leq s_{thres}$.

2. Each gradient update pushed to the servers gets a timestamp t_j indicating the corresponding time from the worker j when the update is performed.

3. A worker i at a given timestamp t_i must have pulled from the server model parameters that have accumulated gradients from every other worker j with timestamp $t_j^{'} \leq t_i - s_{thres} - 1$.

4. The *read-my-writes* property should be satisfied, *i.e.*, a worker will see all previous updates accumulated in the model parameters.

Figure 3.4 fully illustrates the SSP paradigm. Suppose that the three workers start together at the i-th global parameters (w_i) and need to preserve $s_{thres} <= 2$ and worker #2 is stalling compared to the others. After two updates, worker #1 computes its third gradient update but has to wait for worker #2 to pull the next set of model parameters and continue the training. In this way, the bounded staleness condition is satisfied. After worker #2 pushes its gradient update and the server applies it, the pulling process for both workers #1 and #2 will be unblocked and they will both begin their next iteration with parameters w_{i+4}. Since its proposition, SSP has been widely adopted in most of the research works utilizing the parameter server paradigm, either alone [85, 128–132] or by letting the user choose between it and BSP [114, 123] or ASP [58, 59, 123].

Having presented all the basic consistency-related variations of the SGD algorithm, it is worth noting that researchers have also proposed other protocols

based on these variations. Back in 2015, Zhang [133] proposed an alternative of k-batch-ASP, the n-softsync protocol. Having set the parameter n in a setup with l workers, the parameter server will wait to average the first l/n gradient updates it will receive, just like k-batch-ASP. However, the difference is that each one of the l/n gradients will be weighted under a staleness-dependent learning rate. Smaller staleness will provide a larger gradient weight, while if an update occurs from very old model parameters the weight will approximate zero values. One interesting outcome from this research work is that they found that under n-softsync, which lacks synchronization, the model could converge at the same rate as SGD, due to the staleness-dependent learning rate. In the same spirit as Zhang and by performing runtime analysis on the four BSP and ASP variants outlined in Figure 3.3a, Figure 3.3b, Figure 3.3c and Figure 3.3b Dutta [134] also proposed a staleness-dependent learning rate per worker. However, in case the learning rate becomes too slow when dealing with large staleness values, Dutta proposes to use a minimum-value learning rate on such gradients. In the same year, an interesting approach was discussed regarding the framework Litz [135], where they model the training procedure as a task graph providing causal consistency, *i.e.,* only if a task j depends on task i, then i should be executed first compared to j. Moving on 2019, Wang [136] proposed the *overlap synchronous parallel* (OSP) approach. Wang's idea is to preserve two separate threads, one for communication and one for computation purposes, in each worker. Local computations in workers will resume either for a predefined number of iterations or until another model pulls new global model parameters. Local computations will be accumulated in local caches, from where the parameter servers will synchronize at the right time according to the above conditions. Apparently, 2019 was a year when researchers attempted variations of the common consistency protocols, since the *dynamic stale synchronous parallel* [137] (DSSP) was also proposed. DSSP is a variant of SSP with a dynamic bounded staleness threshold. Given beforehand the minimum and maximum possible threshold values, simulations on runtime are used to collect statistics and determine the appropriate threshold value at a given time. Sequentially, MLFabric [138] came out in 2020 where they accelerated BSP, by attempting to minimize the average update transfer time from the workers to the servers. Adopting a shortest-job-first approach, they prioritize the transfer of updates that will be fast, unless an update is about to exceed the staleness value. In that case, this update comes first.

Comparative Effectiveness: BSP is suitable for tasks demanding high accuracy and consistency but suffers from straggler-induced delays. ASP excels in environments prioritizing speed but requires tasks tolerant to inconsistencies. SSP offers a middle ground, combining speed with controlled staleness, but demands careful parameter tuning and monitoring. These trade-offs highlight the importance of tailoring consistency control mechanisms to specific system and application requirements. Consistency control advances are fully summarized in Figure 3.5

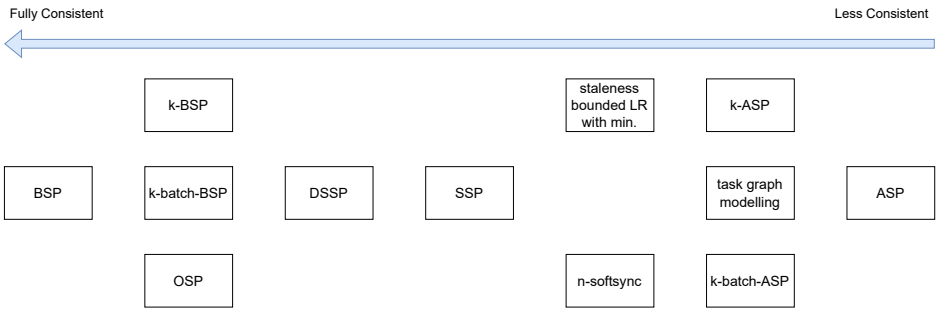


Figure 3.5: Consistency control approaches in the parameter server.

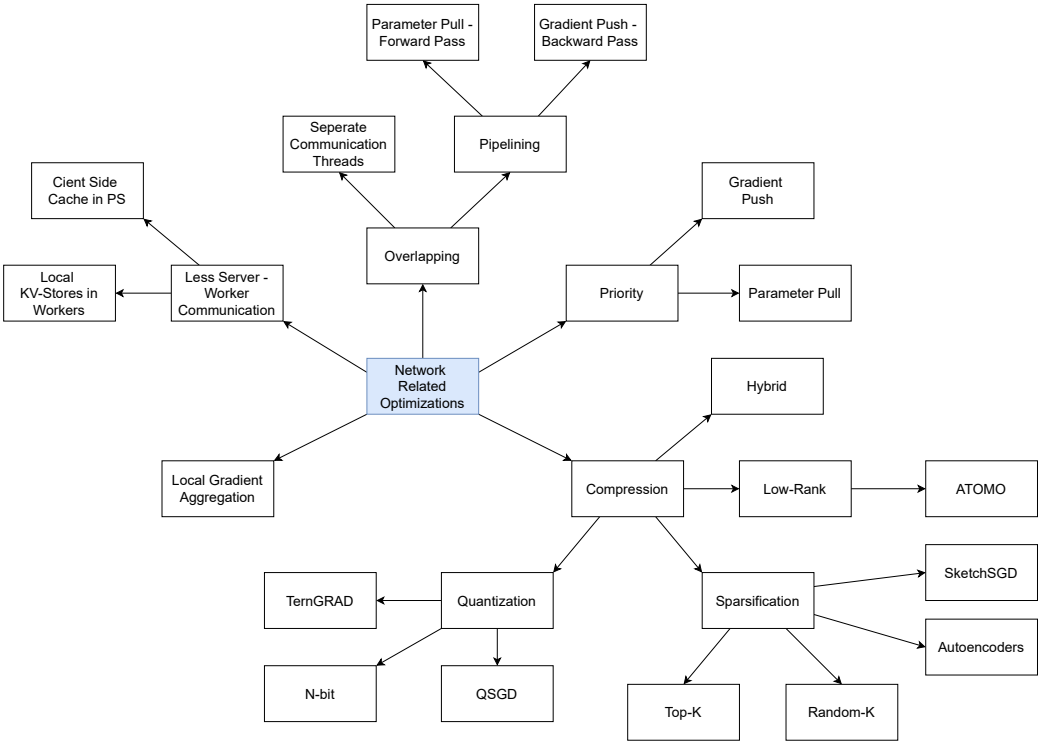


Figure 3.6: Network related optimizations for the parameter server

3.2.3 Network Related Optimizations

Network-related optimizations are critical for addressing the communication bottlenecks inherent in parameter server architectures. As shown in Figure 3.6, the key areas of network optimization include **compression techniques**, **local gradient aggregation**, **prioritization**, **local KV-stores and caching**, and **pipelining and overlapping**. Each approach has its strengths and limitations, which are analyzed below.

Compression techniques such as **sparsification** and **quantization**, are widely used to reduce the size of gradient updates exchanged between workers and servers. Sparse gradients were proposed from the initial research works discussing the parameter server [58, 59], as a solution for gradient compression. KunPeng [130] as a system also proposed the use of sparse representations. In 2018, Wu proposed a gradient compression technique adopting the Top-1% sparsification approach, where only the top 1% of the parameters deemed most important are forwarded to the servers for updates. This method significantly reduces communication overhead while retaining model performance. Similarly, another sparsification technique introduced in 2019 compresses gradients by preserving their original magnitude using a sign operator over groups, enabling training on the ImageNet dataset to complete in 46% less time.

DoubleSqueeze [119] adopts Top-K sparsification, another prominent method, which retains the K largest gradient values during the update process. To further enhance the performance of sparsification methods, researchers have also explored using Top-K sparsification in combination with a low-pass filter, aiming to reduce the noise induced by the sparsification process. An important contribution to this domain is the work by Stich et al. (2018), which introduced the sparsified SGD with error compensation approach [139]. The authors analyze k-sparsification techniques such as top-k or random-k, and demonstrate that by leveraging error compensation mechanisms, where accumulated errors from sparsification are tracked and incorporated into future updates, the convergence rate of sparsified SGD matches that of vanilla SGD. This reduces communication by a factor proportional to the dimensionality of the problem, or even more, without sacrificing convergence speed. Numerical experiments confirm the scalability and efficiency of this method in distributed applications, illustrating its theoretical findings and practical applicability. These advancements demonstrate how sparsification and related techniques, such as error compensation, can significantly reduce communication overhead while maintaining the performance of distributed machine learning systems.

Apart from sparsification, **quantization** is also a well-known approach used for gradient compression. For example, in Bosen [129] 16-bit quantization is used. In 2017, TernGRAD [140] and QSG [141] were proposed. TernGRAD is used to quantize the gradients to ternary precision, while QSGD proposes a lossy stochastic quantization and coding scheme. In 2018, ATOMO [142] was a superclass of TernGRAD and QSGD offering an unbiased estimator using atomic decomposition and sparsification. In the same year, a more complex approach was proposed by Cui [143], who introduced MQGrad. MQGrad utilizes a reinforcement learning model, which is trained in parallel with the network and according to the network's loss it aims to find the optimal bit size that should be used for quantization. 1-bit Quantization scheme is also used by DoubleSqueeze [119], apart from adopting the Top-K sparsification discussed above. Moreover, a more recent example is the idea of quantized preconditions [144]. **Sketching** [145] is an interesting similar approach. In this technique, the server receives gradient sketches from which they

infer the gradients they will use for updating. Another prominent example seems to be the use of autoencoders [146], which are exploited to discover gradient correlations and compress them accordingly. It is worth noting that a wide variety of such algorithms exists. A survey regarding sparsification or quantization algorithms has already been performed in [147] (refer to Table 1 of the aforementioned publication) and therefore this section does not provide a more extensive view of this aspect.

Local gradient aggregation: Besides compressing techniques, it is important to outline further approaches that could achieve better network utilization. Microsoft's project Adam [84] discussed the idea of performing several local gradient aggregations in workers and asynchronously updating the servers using the aggregated values, with similar patterns followed by MLFabric [138].

Prioritization: Bosen [129] assigns priority values, using various techniques, to the computed updates. Prioritizing the update seems to avoid network saturation. The idea of prioritizing is also discussed in [148] where both the pushed updates and the pulled parameter are given priority for transfer according to when they will be reused again, reducing idle time dedicated to communication.

Local KV-Stores and caching: FlexRR [114] examines a different perspective, where the authors *reduce cross-node traffic by including a client-side cache for model parameter entries* in the parameter servers. Cached values contain iteration numbers for easy read access. According to the desired staleness threshold, a read either returns directly or accesses the appropriate server shard to retrieve the correct value. Proceeding on FlexPS [123], this system proposes a stage abstraction for SGD-related algorithms, where a step is considered a stage. To avoid communication between the workers and the server, each worker contains also a local KV store, utilized as a local parameter server, providing *direct memory access*. In this setup, according to which worker will be responsible for the next step, the first worker will forward the model directly to the second, instead of pushing the gradients to the server and then having a pull operation from another worker.

Pipelining and overlapping: Another possible solution includes hiding communication behind computation time. To this end, Poseidon [115] proposes a pipeline where the push operation on gradients of iteration i will be performed in parallel with the gradient computations of iterations $i + 1$. Wang [149] also discusses the pipeline idea, extended to overlap parameter pulling with the forward pass of the training and gradient pushing with the backward pass. The OSP consistency protocol [136], discussed in Section 3.2.2, naturally favors the network in an equivalent way, as communication comes in parallel with computation. It is worth to underline that it also follows the same pattern as project Adam on local gradient aggregation, but also performs accumulations whenever a gradient is pulled.

Parameter servers usually become communication hotspots since all workers need to communicate with this centralized service. The idea of a load balancer seems a natural selection in off-loading hot-spot servers. To this end, the idea of exploiting proxy parameter servers is proposed in Herring [150]. To avoid hot-

spots, there exist also various techniques regarding how the servers shard, store and manage the parameters. However, since parameter management is a large and important topic of discussion, it is presented individually in Section 3.2.4.

Comparative effectiveness: Each of these approaches addresses specific aspects of network optimization, and their effectiveness depends on the workload and system configuration. Compression techniques are most effective in reducing communication overhead but may impact convergence accuracy. Local gradient aggregation and caching are beneficial for tasks with high parameter reuse but require additional resources. Prioritization and pipelining improve communication efficiency but introduce coordination complexity. The choice of optimization strategy should consider the trade-offs between communication reduction, computation overhead, and model convergence. All the ideas and concepts behind network optimizations are summarized in Figure 3.6.

3.2.4 Parameter Management

Parameter servers are usually key-value stores [45, 58, 59, 84, 114, 115, 122, 123, 125–127, 129, 130, 148] in various formats, as DHTs [58], LazyTable [114] or SSPTable [45] (designed for SSP protocol). Another approach proposed by Petuum [85] is the use of a consistent distributed shared memory, which is also utilized in [151]. However, how parameter values are distributed between the available servers can crucially affect the performance due to frequent and concurrent access to them [152], especially as the number of workers increases. Therefore, there has been a large effort over the last years to identify problems in parameter storage and management and propose meaningful solutions.

Parameter storage techniques: Initial parameter storage methods relied on simple hashing techniques to distribute parameters across servers [59]. While this method is straightforward, it fails to account for skewed access patterns, which can lead to imbalanced workloads and reduced performance. For instance, parameters accessed concurrently by many workers create hotspots, slowing down the training process. To address this, Bosen introduced access-pattern-based partitioning, grouping parameters with similar access patterns into the same server [129]. Similarly, in the context of the latent Dirichlet allocation (LDA) algorithm, skewness-aware parameter storage was proposed, distributing parameters based on their usage frequency [151]. While effective for specific applications, these methods require a detailed understanding of parameter access patterns, which may not always be feasible.

Static load balancing: Static load balancing techniques aim to distribute parameters evenly among servers at the start of training. For example, Optimus minimizes imbalances in parameter sizes, update requests, and access patterns through a parameter assignment algorithm that uses a combination of best-fit policies and chunk partitioning [122]. Similarly, FlexPS enables user-defined load balancing policies

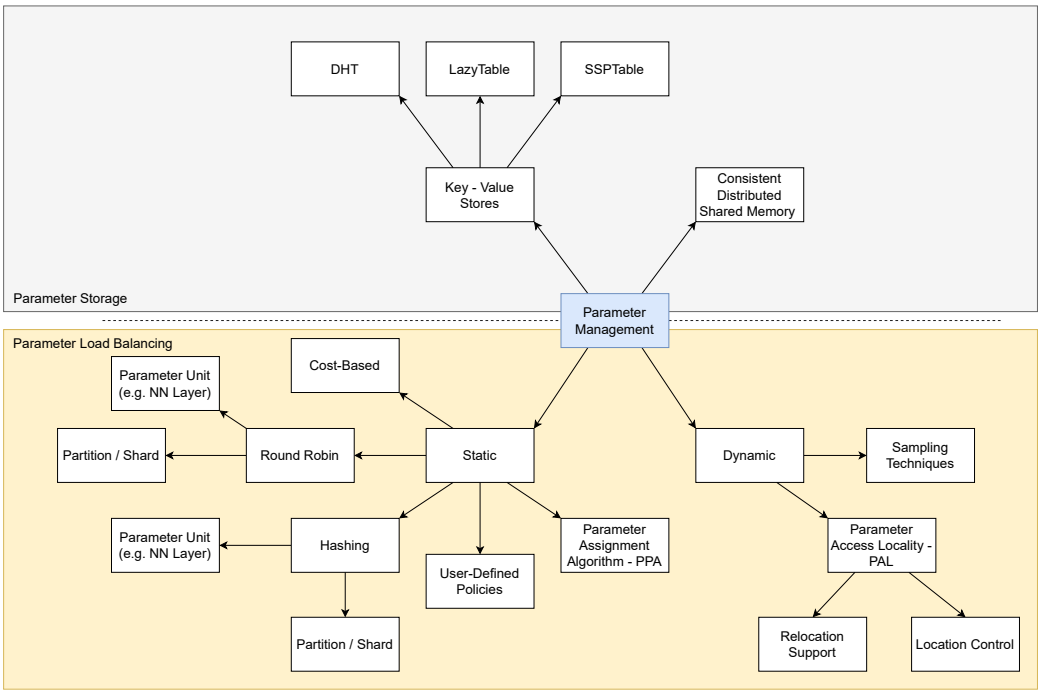


Figure 3.7: Parameter management for the parameter server. Whitesmoke color area (up) presents parameter storage techniques, while the pale orange area (down) indicates load balancing approaches

by providing logs for performance analysis [123]. Other methods, like round-robin assignments of parameter slices [148] or uniform splits across network layers [132], offer simplicity but may fail to adapt to dynamic changes during training.

While the above load balancing approaches are static, Renz discussed in 2020 the idea of dynamically assigning parameters to the server while the model is trained. This idea gave birth to Lapse [125, 126] and its extension Nups [127]. Lapse exploited parameter access locality (PAL) techniques to provide relocation mechanisms and location control. NuPS exploit sampling techniques for specialized applications where different parts of the data distribution need to access different parameters. Dynamic allocation was also proposed by Chen [153] in 2020, who proposed to assign dynamic shards to servers according to skewness and the performance of each available server.

Dynamic parameter management: Dynamic approaches adjust parameter assignments during training to adapt to evolving access patterns. Lapse and its extension NuPS dynamically reassign parameters based on parameter access locality (PAL) and sampling techniques, respectively [125, 127]. These methods are particularly effective in heterogeneous environments where parameter access varies widely across workers. Similarly, Chen proposed dynamically sharding parameters based on server performance and skewness, improving resource utilization and

reducing hotspots [153].

Skewness mitigation via network offloading: To further alleviate the effects of parameter skewness, some approaches offload hot parameter reads and writes from servers to programmable network switches [152]. This method leverages specialized hardware to handle frequently accessed parameters, reducing server load and improving overall throughput. However, the deployment of such specialized hardware introduces additional costs and complexity, limiting its applicability to well-funded, large-scale systems.

Comparative effectiveness: The choice of parameter management technique depends on the workload and system configuration. Static load balancing approaches, such as round-robin or uniform splits, are simple and effective for homogeneous environments but struggle with dynamic workloads or skewed access patterns. Dynamic methods, like Lapse and NuPS, excel in heterogeneous and high-skew scenarios but require additional computation and monitoring. Network offloading offers significant performance gains in handling hot parameters but may not be feasible for smaller or resource-constrained setups. As shown in Figure 3.7, parameter management strategies are categorized into two key areas: storage techniques and load balancing approaches. The figure highlights how innovations like skewness-aware partitioning and dynamic reassignment complement traditional methods, offering a comprehensive view of the solutions available for parameter server optimization.

3.2.5 Straggler Problem

In cloud environments, the term *stragglers* refers to tasks stalling other tasks [154]. In the parameter server context, a task is the iteration part of some iterative optimization algorithm, executed on a worker. A stalling task could also exist in the servers, during updating or pulling the global parameters.

An interesting classification of stragglers is provided in [155]. In this paper, they distinguish straggler appearance as random and deterministic. Random stragglers may occur in cases of temporary events, such as garbage collection or OS-related tasks. To the contrary, deterministic stragglers are attributed to machine heterogeneity. Therefore, there are works that either cover the topic of stragglers in general or propose specialized solutions for heterogeneous environments.

Systems implementing the SSP protocol (discussed in Section 3.2.2) provide some general control over stragglers. Compared to BSP, the performance is partially hurt, since the fast worker will wait for the straggler only upon the staleness threshold. In contrast to ASP, SSP will ensure convergence due to bounded staleness. [45, 156]. However, apart from the SSP protocol, further optimizations have been proposed to mitigate the straggler effect.

For instance, a **stop epoch earlier** approach is followed in project Adam [84], the authors propose to consider an epoch completed on stragglers when they have performed 75% of the epoch training. They claim to preserve the same levels of

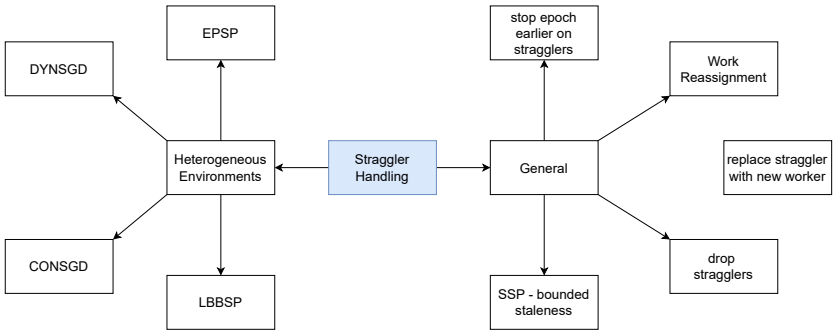


Figure 3.8: Straggler handling approaches in the parameter server

accuracy in resulting models with up to $20X$ speedup.

Work stealing / reassignment is also a prominent approach discussed FlexRR in [114]. In this case, workers can self-identify whether they are stalling. If this happens, they offload work to the faster trainers.

Focused on GPU environments, Poseidon [115] simply chooses to **drop stragglers** when they occur, since the authors claim do not want to hurt the performance capabilities offered by accelerators. Furthermore, Optimus [122] finds stragglers using statistics and replaces the worker with a new one. Recently, Li *et al.* proposed the Sync-Switch approach, which offers an online version changing from BSP to ASP when stragglers are detected.

While the above optimizations are general and could be applied regardless of the homo- or the heterogeneity of the cluster, environments with non-uniform hardware types could benefit from further optimizations. The most prominent solution to exploit all workers in heterogeneous environments seems to appropriately adapt SGD [116, 117, 155, 157, 158] through **dynamic adjustment of learning rate, batch size and parallelism**. Jiang *et al.* proposed two variants of SSP, the *CONSGD* and *DYNSGD*, which do learning learning rate assumptions, to smooth stragglers. *CONSGD* proposes a global learning rate, while *DYNSGD* considers heterogeneity via staleness to design a per worker learning rate. Zhou *et al.* in Falcon [116, 117] discussed the idea of adapting BSP and SSP into an elastic-parallelism synchronous parallel, which adapts the worker parallelism level according to straggler appearance and reassigns task from stragglers to fast workers. In 2019, Yu *et al.* [158] proposed to adjust the batch size per worker to handle stragglers, by giving them less work. A year later, Chen *et al.* proposed LBBSP following the same paradigm, but with different protocols on CPU and GPU clusters.

Comparative effectiveness: The effectiveness of these approaches depends heavily on the system configuration and workload. Work reassignment, elastic parallelism, and batch size adjustments are particularly effective in heterogeneous environments, where they balance the workload dynamically. Straggler dropping, while straightforward and efficient, is better suited for homogeneous systems with

high-performance workers. Approaches like dynamic learning rate adjustment offer a middle ground but may slow convergence under extreme straggler scenarios. Figure 3.8 recaps this section, outlining all the techniques mentioned.

3.2.6 Fault tolerance

Fault tolerance refers to how capable the parameter server setup is on handling possible node failures, either on the training or the inference process. Since training neural networks is a very time consuming task and in distributed systems one should consider the case of node failure, one research topic in such setups refers to how one could resume the training in case of failures. As discussed in Section 3.1, the parameter server in asynchronous mode is by default more tolerant in node failures compared to synchronous distributed setups. However, this section surveys proposed approaches for fault tolerance in the parameter server regardless of the training mode.

Since fault tolerance is an important factor for distributed systems and architectures [159–161], multiple researchers have proposed ways of improving the tolerance of the parameter server over the years. Influenced by other works starting to propose the parameter server architecture, Carnegie Mellon University researchers cooperated with Google's Smola, who proposed the general architecture for parallel models (Section 3.1), to specialize the parameter server for distributed machine learning in training mode [58, 59]. Among other optimizations, they discuss the fault tolerance issue, where they propose to store model parameters in distributed hash tables replicated in more than one server, guaranteeing fast recovery in case of node failures.

A year later in 2014, Microsoft launched project Adam [84] as their distributed deep learning system, which is also based on the parameter server paradigm. Regarding fault-tolerance. Microsoft adopts the paxos algorithm in a cluster of controller machines. In further detail, parameter shards are **replicated between the servers**, which send periodic heartbeats to the controllers. In case of failure, controllers relocate the latest shards to live replicas. Later on 2017 [130], Alibaba designed and implemented *KunPeng*, which is a parameter-based system, with extended fault-tolerant mechanisms compared to *Petuum* for industrial environments where multiple job types are executed. *KunPeng*'s servers back up their parameter shards in a distributed file system and simultaneously cache them in memory for quick recovery in case of a failover.

Periodic checkpoints have been discussed in various manuscripts in the years following the above research works. For instance, Poseidon [115] encapsulated this technique to handle failures. In 2018, Optimus [122] and FlexPS [123] have been proposed, where they also use checkpoints to resume training in case a node crashes, with the former performing the checkpoints in an external system and the latter exploiting user-defined policies in performing checkpoints. According

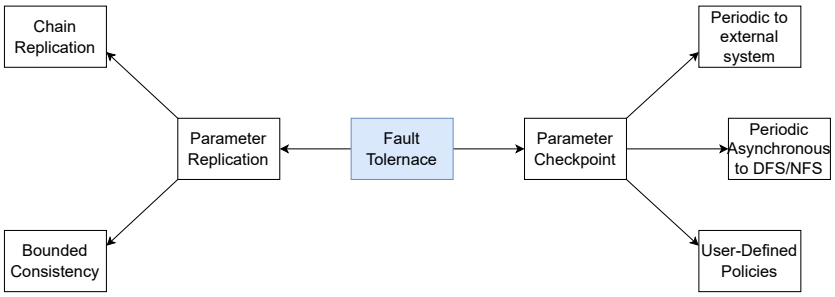


Figure 3.9: Fault tolerance approaches in the parameter server

to Alibaba researchers [131], *Petuum* [85] was proposed by their team in 2015 following the design of a communication efficient parameter server [128]. While these papers claim the importance of fault tolerance, *Petuum* creators argue the need for fine-grain fault tolerance, since in terms of machine learning optimization fast convergence is needed to be ensured. Therefore, they offer reliability in terms of system failure with often model checkpoints, by continuing the training process from the last checkpoint available.

However, apart from these baseline techniques, more **advanced fault-tolerant techniques** have been discussed. An interesting approach referring to parameter server's straggler problem is FlexRR [114], where they propose an algorithm named **approximate snapshot** to support training after failures. **Approximate Snapshot** will lead servers to checkpoint data at a specific timestamp after they have received the corresponding clock-related messages from all workers. Parameters written on checkpoints are approximate, since due to the SSP consistency model FlexRR offers, they may have been updated by future gradients with respect to the checkpoint time. Moving on in 2020, MLFabric [138] has alternate base chain data replication discussed earlier, with discussing **bounded consistency** between one parameter server and another containing its replicas. Since they built MLFabric for multi-tenant environments, the author addresses that it is important to find a tradeoff between the bandwidth and time consumed in chain replication and the quality of the parameter backups, and, therefore propose bounded consistency between the original parameter and the replica values.

Comparative effectiveness: Parameter replication offers high reliability but comes with heavy resource demands, making it suitable for critical systems. Checkpointing provides a balance of reliability and performance but may be less effective for systems with frequent failures. Advanced techniques like approximate snapshotting and bounded consistency are resource-efficient but require careful implementation to avoid compromising accuracy. In Figure 3.9, all the above are summarized in a diagram outlining the various techniques proposed related to fault tolerance issue.

3.3 Discussion

The parameter server architecture has emerged as a foundational framework in distributed machine learning, enabling scalable and efficient training for modern applications. Over the years, various optimizations have been proposed to address challenges in consistency control, network utilization, parameter management, straggler handling, and fault tolerance. These optimizations not only showcase practical effectiveness but also highlight the metrics used to evaluate their performance.

Consistency control techniques like stale synchronous parallel (SSP) and dynamic stale synchronous parallel (DSSP) balance staleness and throughput while maintaining accuracy. Performance evaluation often includes metrics such as training convergence time, throughput (iterations per second), and model accuracy. For example, DSSP dynamically adjusts staleness thresholds, improving throughput without degrading model performance [137].

Network optimizations are commonly evaluated using metrics such as bandwidth utilization, communication overhead, and time to convergence. DoubleSqueeze, through Top-K sparsification, reduced ImageNet training time by 46%, demonstrating significant communication efficiency [119]. Similarly, TernGRAD's quantization methods minimized bandwidth usage while maintaining accuracy levels comparable to uncompressed training [140].

Parameter management solutions, such as Optimus and NuPS, are assessed using metrics like load imbalance, latency, and system throughput. Optimus achieved even workload distribution across servers, minimizing latency and improving overall throughput in heterogeneous environments [122]. NuPS dynamically assigned parameters during training, addressing skewed workloads and optimizing resource utilization [127].

Straggler handling approaches utilize metrics such as task completion time, worker utilization, and system efficiency. elastic-parallelism synchronous parallel (EPSP) demonstrated reduced delays caused by slow workers, redistributing workloads dynamically to enhance system efficiency [116]. Batch size adjustments, as seen in LBBSP, further improved straggler performance by adapting workloads based on worker capabilities [155].

Fault tolerance mechanisms, like checkpointing in KunPeng and MLFabric's bounded consistency replication, are evaluated using metrics such as recovery time, checkpointing overhead, and training robustness. KunPeng's parameter backups ensured seamless recovery during node failures, minimizing downtime and maintaining training continuity [130].

These metrics provide a comprehensive framework to assess the practical effectiveness of parameter server optimizations. By examining factors such as throughput, communication efficiency, workload distribution, and fault recovery, these techniques demonstrate measurable improvements across diverse distributed learn-

Table 3.3: Surveys on parameter server architecture.

Ref.	Year	Title	Focus Area	Limitations
[103]	2016	*Survey of scaling platforms for deep neural networks*	Mentions parameter server as a method for scaling CPUs for learning workloads, among other accelerators like GPU clusters or quantum computing.	Only a brief mention; lacks discussion of synchronization modes, communication optimization, or hyperparameter tuning.
[104]	2016	*Parallel processing systems for big data: A survey*	Discusses big data processing systems, mentioning parameter server adoption by Petuum and describing its basic capabilities.	Focuses on big data processing; lacks in-depth discussion of parameter server-specific synchronization protocols or distributed learning strategies.
[162]	2018	*A Quick survey on large scale distributed deep learning systems*	Reviews distributed deep learning approaches, including parameter server architecture. Discusses basic synchronization modes (BSP and ASP) and hyperparameter considerations (e.g., batch size and learning rate).	Provides limited details on advanced synchronization strategies, communication optimizations, or scalability challenges.
[105]	2020	*A survey on distributed machine learning*	General survey on distributed machine learning. For parameter server, covers synchronization modes (BSP, ASP, stale synchronization) and their trade-offs in terms of model consistency and communication overheads.	Lacks coverage of hyperparameter tuning, worker-server interaction optimization, or model parallelism techniques.
[106]	2021	*A quantitative survey of communication optimizations in distributed deep learning*	Explores communication optimizations in distributed training, focusing on parameter partitioning, fusion, and scheduling within a parameter server network.	Narrow focus on communication aspects; does not address synchronization or hyperparameter adaptation in depth.
[107]	2021	*Communication optimization strategies for distributed deep neural network training: A survey*	Discusses communication optimizations, including gradient compression techniques (e.g., sparsification, quantization) and overlapping computation and communication.	Focuses exclusively on communication optimization; lacks coverage of synchronization protocols, scalability, or model-parallel training considerations.

ing systems. The performance benchmarks, including faster convergence times and improved resource utilization, highlight their practicality in both academic and industrial applications. But why was it important to perform such an extensive survey in the context of this thesis? While researching the parameter server domain, there was no existing survey that comprehensively covers the *Parameter Server* architecture. Existing surveys cover specific parts of the *Parameter Server* architecture, and are briefly presented in Table 3.3

In 2016, Ratnaparkhi [103] examined scaling techniques for deep learning (mentioning *Parameter Server* only tangentially), and Zhang *et al.* [104] listed the *Parameter Server* as one of many big data systems adopted by Petuum. A concise overview by Zhang *et al.* [162] described BSP and ASP synchronisation in the *Parameter Server*, while Verbraeken *et al.* [105] analysed trade-offs around stale

synchronisation. More recently, Shi *et al.* [106] studied communication optimizations as the number of server shards grows, and Ouyang *et al.* [107] focused on gradient compression and compute/communication overlap.

Although these works illuminate individual facets—communication, synchronisation, or partitioning—they lack a holistic view. Chapter 3's survey fills that gap by systematically covering *Parameter Server* synchronisation strategies (BSP, ASP, SSP), hyper-parameter adaptation, worker–server interaction, and model-scalability issues, while drawing practical lessons from TensorFlow, MXNet, DeepSpeed, Ray, and Horovod.

3.3.1 Exclusion of federating learning

Parameter-server architectures and federated learning (*FL*) both employ distributed computation, yet their goals diverge sharply. As described by McMahan *et al.* [163], *cross-silo* FL targets privacy-sensitive, non-IID, and often unbalanced data across thousands of devices, emphasising communication efficiency via model-parameter exchange. In contrast, the *Parameter Server* assumes IID, balanced shards and relies on gradient exchange for rapid convergence in cluster environments.

Modern FL research spans communication efficiency [164–166], scalability [167–169], personalisation [170–172], and security & privacy [173–175]. Several recent surveys [176–188] offer deep dives into one or more of these aspects. Because this thesis focuses on the *internal* mechanics of the *Parameter Server*, *FL* is treated as a distinct paradigm and its specialised challenges are left outside the scope of this survey and the thesis overall.

3.4 Focus on Consistency Control

The survey provided in this chapter has given a deeper understanding of the *Parameter Server* architecture. However, the research question set to be answered in this thesis (Section 1.2) was related to optimizing the asynchronous learning in the parameter server architecture. In this direction, *Sync-Switch* [189] proposed to initially perform some training epochs under BSP to initialize model weights near the local minimum and then finalize the process faster using ASP for the remaining epochs. *Sync-Switch* discovers its switching point from BSP to ASP via an *offline* binary search across multiple runs.

Considering all the analysis and the findings from the survey in Section 3.2, BSP usually comes with network hotspots, due to the synchronous gradient pushes to centralized servers. For synchronous training, *All-Reduce* seems to be a more prominent approach. Recent work in both *All-Reduce* and *Parameter Server* training explicitly balances consistency and runtime. For example, Prague [190] transfers the async semantics of AD-PSGD [191] to an *All-Reduce* setting, while SSP [45] and its adaptive variant DSSP [137] bound staleness in the *Parameter Server*.

Worker-selection or pruning mechanisms—fedPAGE [192], FEDL [193], PFL [194], PruneFL [195]—further reduce straggler impact with minimal accuracy loss.

Apart from architectural and consistency policies, data-related policies are also widely used to help the training process in relaxed training approaches, like ASP. As discussed in Section 3.2, Damaskinos *et al.* [196] apply a staleness-decay factor within SGD; Dutta *et al.* [134] and Huang *et al.* [123] vary learning rate or exploit ageing parameters; Viswanathan *et al.* [197] propose MLfabric, a communication layer that accelerates gradient flow and handles stragglers. Apart from these advances, related to weighting parameters and gradients differently, the concept of stratification is also widely used in learning concepts. For example, it is long used to reduce variance in single-node cross-validation [198] and re-appears in domain-specific models [199], distributed graph partitioning [200], and data-sensitive hashing [201]. In single-node contexts, Polyzotis *et al.* [202] flag data skew as a source of bias, while Hsieh *et al.* [203] propose advanced generation techniques when data shards are physically disjoint.

Inspired by all the above contributions and the overall findings and understanding of the parameter server provided through this extensive survey, the rest of this thesis will focus on two different approaches that aim to help ASP. Chapter 4 will discuss *Strategy-Switch* that follows the paradigm of *Sync-Switch*, by replacing BSP *Parameter Server* with *All-Reduce* approaches. Furthermore, instead of using an offline rule for the switching point, *Strategy-Switch* extends this line by demonstrating that an inexpensive online switch can match the accuracy of fully synchronous training while approaching the speed of ASP. Especially, under heterogeneous clusters with constant stragglers, *Sync-Switch* would default to BSP *Parameter Server*, while the *Strategy-Switch* will exploit ASP in any case. The second approach that aims to aid the training process under ASP *Parameter Server* follows data-related policies and specifically stratification techniques, as in the novel works presented above. Specifically, in Chapters 5 and 6 this thesis is the first to examine whether *data sharding* can stabilize learning in an asynchronous *Parameter Server*. Prior work tackles other ASP challenges, while in this thesis a shared file system is leveraged to deploy simpler—but effective—stratification schemes, yielding up to 8× lower training-loss variance.

Chapter 4

Synergizing All-Reduce and Asynchronous Parameter Server via Strategy-Switch for Training Efficiency

Chapter 4 classified PS optimisations into five domains (Consistency Control, Network Optimization, Parameter Management, Straggler Problem and Fault Torelance). This chapter picks up the thread by targeting the first two domains — Consistency Control and Straggler mitigation — through a hybrid approach called Strategy-Switch.

The preceding survey in Chapter 3 highlighted the central role of *consistency control* in balancing these trade-offs. In particular:

- **Bulk Synchronous Parallel (BSP)** enforces full synchronization each iteration, ensuring model consistency but suffering from straggler-induced stalls [45, 90].
- **Asynchronous Parallel (ASP)** removes synchronization barriers, accelerating training but introducing unbounded parameter staleness [109].

Moreover, the survey provided in Chapter 3 showed that synchronous Parameter Server (i.e. BSP-PS) often incurs higher network hotspots and server contention, due to all workers pushing gradients to a central server. In contrast, another synchronous data parallel paradigm, named ***All-Reduce*** leverages decentralized ring or tree aggregation, avoiding a single point of congestion and yielding up to 20–30% lower network traffic under BSP, as shown through benchmarking (Section 4.2, Figure 4.4). This makes All-Reduce the preferred BSP strategy to minimize communication overhead.

Building on these insights, and inspired by Sync-Switch [189], ***Strategy-Switch*** is introduced, a hybrid framework that:

1. *Starts* with All-Reduce to reliably guide the model toward a strong optimum;

2. *Automatically switches* to asynchronous Parameter Server once the training dynamics stabilize, governed by an empirical rule based on validation-loss fluctuations (Section 4.3.3).

The key points of this Chapter are the following:

- A benchmarking analysis comparing convergence and performance of All-Reduce vs. asynchronous PS (Section 4.2).

- The design and evaluation of the *Strategy-Switch* algorithm on standard image-classification benchmarks (Section 4.4).

- An empirical transition rule that optimally determines the switch point (Section 4.3.3).

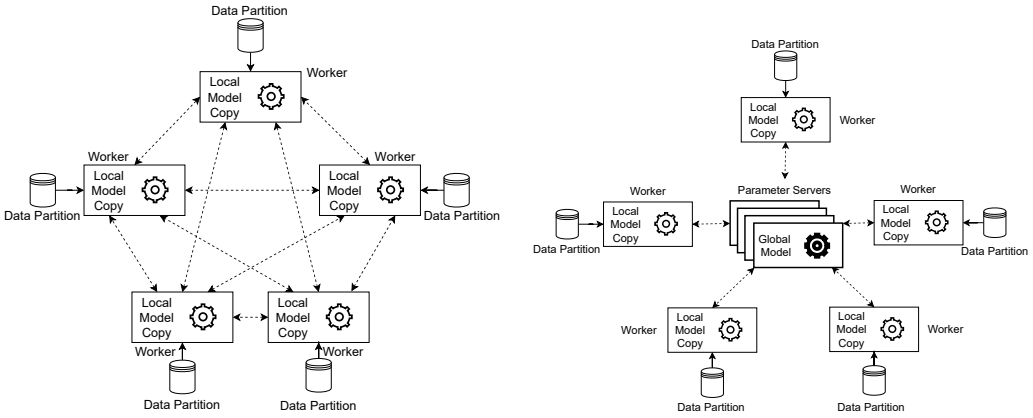
The remainder of this chapter is organized as follows: Section 4.1 briefly recalls the key properties of All-Reduce and Parameter Server; Section 4.3 details the *Strategy-Switch* methodology; Section 4.4 evaluates its performance and transition rule.

4.1 Theoretical Background

In this section, the necessary background on the two distributed training paradigms is established, *i.e.* the *All-Reduce* and *Parameter Server* integrated into the *Strategy-Switch* training methodology. *Parameter Server* is thoroughly discussed in Chapter 2, but is also summarized in this section for sake of completeness.

4.1.1 All-Reduce

All-Reduce [81, 83, 204] embodies a decentralized training methodology where all workers engage in gradient exchange. This process is illustrated in Figure 4.1a. At the onset of a training step, each worker extracts a mini-batch from their local data partition, where they compute gradient vectors. Synchronous all-reduce techniques facilitate the exchange and aggregation of these gradients among workers and subsequently update each worker's local model for the next iteration. This synchronous nature ensures uniformity across local models when the next iteration is initiated. A widely adopted *All-Reduce* approach within various learning systems like TensorFlow [63] is the Ring All-Reduce [82] method.



(a) *All-Reduce* (b) *Parameter Server*

Figure 4.1: Distributed training architectures: (a) All-Reduce; (b) Parameter Server. Dashed lines indicate communication, while solid lines indicate data extraction.

4.1.2 Parameter Server

Parameter Server [39, 57–59] constitutes a centralized approach for distributed model training. In this setup, servers maintain a global model, whereas workers compute gradients on local model copies, using data from a data partition assigned to them, as depicted in Fig. 4.1b. *Parameter Server* training can operate either in bulk synchronous (BSP) or asynchronous parallel (ASP) mode. During a training step, workers retrieve the latest model parameters from servers, compute gradients using their local data, and push these gradients back to update the global model on the servers. In ASP, servers update the global model upon receiving new gradients, whereas in BSP, they wait for gradients from all workers. Throughout this paper, any reference to *Parameter Server* without specifying the training mode refers to ASP. Although ASP is faster since it lacks synchronization-related performance overheads, it may suffer from stale gradients, potentially compromising model accuracy.

4.1.3 Hyperparameters in Distributed Settings

In these training paradigms, hyperparameters such as mini-batch size and learning rate are adjusted to emulate equivalent single-node setups [205]. To mirror single-node training, the global batch size should match the single-node counterpart, resulting in a scaled per-worker size based on the worker count. In asynchronous *Parameter Server*, the learning rate is also adjusted since each worker contributes independently to the global model located on the servers.

Table 4.1: Benchmark Overview

Benchmark	Dataset	Network	#Epochs	Hyperparameters
#B1	CIFAR-10	ResNet-20v1 (0.27M params)	182	SGD, mini-batch size 128, momentum 0.9, step decay
#B2	CIFAR-100	ResNet-32v1 (0.48M params)	200	LR initial 0.1, divided by 0.1 at epochs 82 and 133

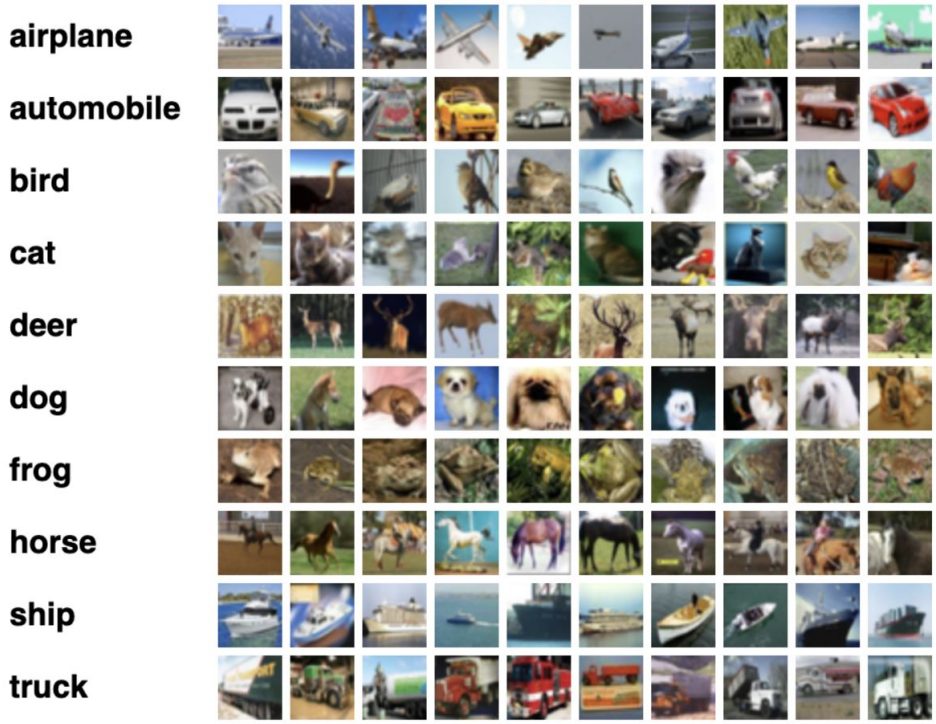


Figure 4.2: Examples of CIFAR-10 images for each category.

4.2 Benchmarking baseline strategies (network & straggler sensitivity)

This section presents an experimental evaluation of training using the *All-Reduce* and *Parameter Server* paradigms to delve deeper into their characteristics.

4.2.1 Experimental Setup

The evaluation involved two benchmarks utilizing CIFAR datasets [206] and ResNet-based neural networks [13] as outlined in Table 4.1.

CIFAR-10 [206] is a set of labeled images. The CIFAR-10 dataset consists of 60000 32x32 color images divided into 10 classes, with 6000 images per class. The

Table 4.2: VM Specifications in Clusters

VM Flavor	vCPUs	RAM	HDD Storage
m1.xxlarge1	8	32GB	100GB
m1.large3	4	16GB	60GB
regale-small	2	8GB	30GB

Table 4.3: Clusters used for Experiments

Cluster Code	Cluster Type	m1.xxlarge1	m1.large3	regale-small
#C1	Homogeneous	5	-	-
#C2	Heterogeneous	5	2	2

dataset is divided into 50000 training data and 10000 test data. The dataset is divided into five training batches and one control batch, each of which consists of 10000 images. The control batch contains exactly 1000 randomly selected images from each class. Training batches contain the remaining images in random order, but some training batches may contain more images from the same class. In total the training batches contain exactly 5000 images of each class. The classes into which the data is divided are: plane, car, bird, cat, deer, dog, frog, horse, ship, and truck. In Figure 4.2 examples of images from each class[1] can be found. The CIFAR-100 dataset is similar to CIFAR-10, but it contains 100 classes with 600 images each. For each class, there are 500 training images and 100 testing images. The 100 classes in CIFAR-100 are organized into 20 superclasses. Each image has both a "fine" label, indicating its specific class, and a "coarse" label, representing its superclass.

The hyperparameters referenced in Table 4.1 are adjusted for distributed setups, as explained in Section 4.1.3. Virtual machines from a private Openstack cloud cluster are used to setup the evaluation infrastructure. The infrastructure consists of a homogeneous cluster (i.e., the cluster nodes hardware configuration is the same) and a heterogeneous cluster (i.e., the cluster nodes hardware configuration varies between the nodes). Table 4.2 details the specifications of the Openstack cluster virtual machine flavors utilized in the clusters. In Table 4.3 an overview of the clusters utilized in the experiments is provided. TensorFlow v2.6.2 for both *All-Reduce* and asynchronous *Parameter Server* training is employed in each cluster. Runs are capped at 182 and 200 epochs, the standard horizon for ResNet on CIFAR-10/100 [13].

[1]downloaded from https://www.cs.toronto.edu/ kriz/cifar.html

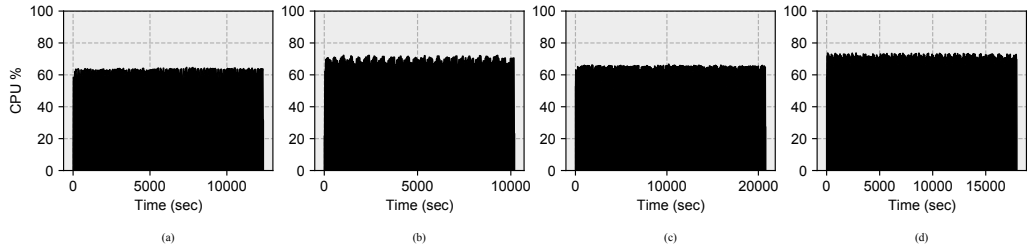


Figure 4.3: #C1 Cluster CPU Utilization for training benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server*

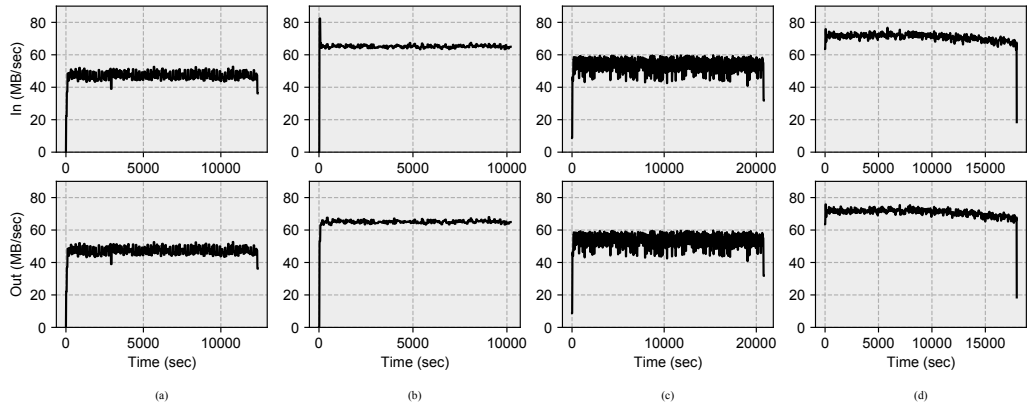


Figure 4.4: #C1 Cluster Network Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server*

4.2.2 Homogeneous #C1 Cluster Benchmarking

Figure 4.3 illustrates the CPU utilization of *All-Reduce* and *Parameter Server* training in the homogeneous #C1 cluster. For both benchmarks, #B1 and #B2, *All-Reduce* and *Parameter Server* display similar CPU utilization in this environment. *All-Reduce* hovers around 65%, while *Parameter Server* peaks at approximately 75%. The *All-Reduce* strategy exhibits lower performance due to synchronization overheads, whereas *Parameter Server* demonstrates improved cluster utilization as workers update the model independently.

The network traffic in #C1 (Figure 4.4) confirms similar levels of incoming and outgoing traffic for *All-Reduce* across both benchmarks, approximately 50 MB/sec. In contrast, *Parameter Server* manifests higher network usage with approximately 65 MB/sec for #B1 and 70 MB/sec for #B2. This disparity results from the increased complexity of the ResNet-32 model used in #B2, leading to a greater exchange of gradients and model parameters.

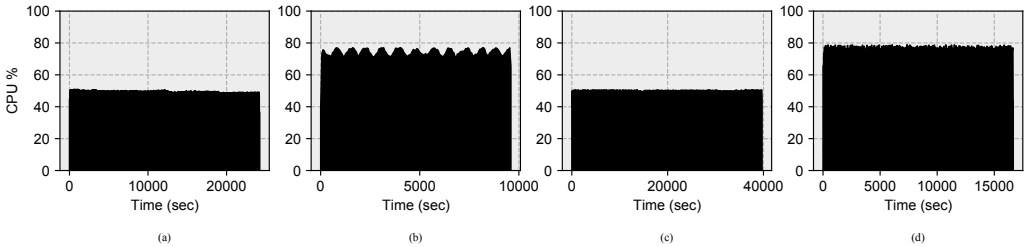


Figure 4.5: #C2 Cluster CPU Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server*

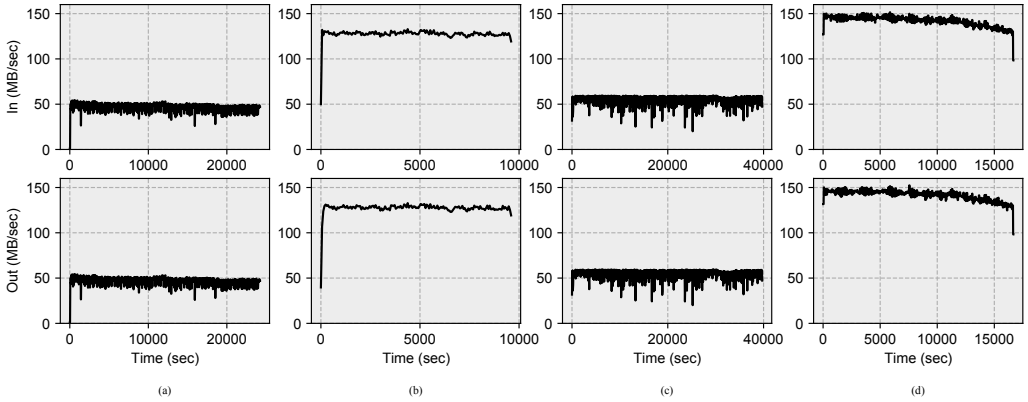


Figure 4.6: #C2 Cluster Network Utilization for training the benchmarks under *All-Reduce* and *Parameter Server* (a) #B1 -*All-Reduce* (b) #B1 -*Parameter Server* (c) #B2 -*All-Reduce* (d) #B2 -*Parameter Server*

4.2.3 Heterogeneous #C2 Cluster Benchmarking

Figure 4.5 displays the CPU utilization of *All-Reduce* and *Parameter Server* training in the heterogeneous #C2 cluster. Compared to the homogeneous cluster, *All-Reduce* and *Parameter Server* showcase significantly divergent CPU utilization. *All-Reduce* demonstrates lower CPU usage at $\sim 50\%$ for both benchmarks, while *Parameter Server* exhibits much higher usage ($\sim 80\%$). The synchronous nature of *All-Reduce* results in varied worker speeds, impacting performance significantly in the heterogeneous environment. This impact is due to the different computing capabilities of the participating nodes and the requirement for synchronization. The synchronization requirement stalls faster nodes that need to wait for the slower nodes computation to finish. Conversely, *Parameter Server*, operating asynchronously, presents consistent utilization irrespective of worker differences, making it more adaptable to cluster heterogeneity. This utilization consistency is the result of the asynchronous computation: faster nodes do not need to wait for

slower nodes to finish their computations before gradient updates. Nevertheless, this speed comes at the expense of slower convergence, since "stale" computations of slower nodes diverge the gradient computation away from its correct value by adding noise at every computation step.

In terms of network traffic (Figure 4.6), *All-Reduce* exhibits similar levels compared to the homogeneous cluster, while *Parameter Server* doubles the network traffic on #C2 compared to #C1. The higher traffic validates the effect of stale value updates of slower nodes, as more message exchanges (therefore more network traffic) between nodes are needed in order to come to the same conclusion with the homogeneous setup.

4.3 Strategy-Switch: adaptive consistency controller

In this section, a hybrid train strategy for distributed training, *i.e.* the *Strategy-Switch*, is proposed. An extensive discussion on the design of *Strategy-Switch* is provided and finally an empirical rule on the switching point is provided.

4.3.1 Approach

In order to combine both the benefits of the two distributed training setups discussed in Section 4.1, *Strategy-Switch* is proposed. *Strategy-Switch* is a hybrid distributed setup, which performs the model training in an *All-Reduce* approach up to a specific epoch and then proceeds with training under an asynchronous *Parameter Server* setup. *Strategy-Switch-*α% is illustrated in Algorithm 4.7, where α% represents the percentage of epochs performing *All-Reduce* training.

After the completion of the last *All-Reduce* epoch, *Strategy-Switch* dumps (saves) the global model on a Distributed File System (DFS), so that the parameter servers (PS) can load it as the initialization point for the subsequent asynchronous training stage. In practice, this model exchange procedure requires minimal overhead, as the corresponding checkpoint is copied to the DFS and then broadcast by the servers.

4.3.2 Theoretical Explanation

Convergence in Strategy-Switch

In *Sync-Switch* [189], the authors provide a detailed theoretical explanation on why changing training in the *Parameter Server* architecture from BSP mode to ASP could benefit the training process (Section IV-A). In the proposed *Strategy-Switch*, the BSP *Parameter Server* phase is replaced with an *All-Reduce* phase.

```
1: procedure StrategySwitch-α%(m, d, hp, e, α)
2:                         ▷ m: trainable model
3:                         ▷ d: training and validation data
4:                         ▷ hp: global hyperparameters
5:                         ▷ e: total number of epochs
6:                         ▷ α: percentage of epochs under All-Reduce
7:     Deploy set of All-Reduce workers: w_allred
8:     for i = 1 to α × e do
9:         train(w_allred, m, d, hp, i)
10:    end for
11:    Dump model m on DFS
12:    Initiate parameter servers: ps
13:    ps.load(m)
14:    Deploy set of Parameter Server workers: w_ps
15:    for i = α × e + 1 to e do
16:        train(w_ps, ps, m, d, hp, i)
17:    end for
18:    return trained model m
19: end procedure
```

Figure 4.7: StrategySwitch-α%

Therefore, in *Strategy-Switch*, *All-Reduce* training can be considered as a stable approach to lead the model parameters closer to the optimization point, similar to the approach of the *Sync-Switch* [189]. When gradients are small, smaller movements to the model parameters are caused. Hence, the model is not crucially altered and the small change of the model parameters render it less vulnerable to stale gradients that may occur in asynchronous learning setups.

Why *All-Reduce* over BSP *Parameter Server*

As shown in Section 4.2, asynchronous *Parameter Server* consumes more network compared to *All-Reduce*. However, the higher CPU utilization attributed to the lack of synchronization leads to faster training. When using BSP *Parameter Server*, the synchronization will impact cluster utilization and the training will also be prone to possible network bottlenecks. Such bottlenecks may be attributed to either server hotspots [207] and network waiting time for synchronization [45]. Therefore, for synchronous training, it is more efficient to exploit *All-Reduce* instead of BSP *Parameter Server*, since network hotspots will be avoided, due to *All-Reduce* being decentralized. Various studies [208–210] discussing the network efficiency of *All-Reduce* compared to *Parameter Server* support this claim.

4.3.3 Empirical rule for the switching point

1: **procedure** EMPIRICAL RULE STRATEGYSWITCH(m, d, hp, e)
2: ▷ m: trainable model
3: ▷ d: training and validation data
4: ▷ hp: global hyperparameters
5: ▷ e: total number of epochs
6: Deploy set of *All-Reduce* workers: **w_allred**
7: val_window = rolling_window(6)
8: $s = +\infty$
9: $i = 0$
10: **while** $s > 1\%$ **do**
11: train(**w_allred**, m, d, hp, i)
12: roll(val_window, epoch_val_loss)
13: $s = \frac{\sum_{j=0}^{4} \|v_{loss}(i-j) - v_{loss}(i-j-1)\| \times 100\%}{5 \times v_{loss}(i-j-1)}$
14: $i = i + 1$
15: **end while**
16: start_epoch_ps = i
17: Dump model m on DFS
18: Initiate parameter servers: **ps**
19: **ps**.load(m)
20: Deploy set of *Parameter Server* workers: **w_ps**
21: **for** $i = start\_epoch\_ps$ **to** e **do**
22: train(**w_ps**, **ps**, m, d, hp, i)
23: **end for**
24: **return** trained model m
25: **end procedure**

Figure 4.8: Empirical Rule StrategySwitch

The design of *Strategy-Switch* raises one important question: When is it the right time to change from *All-Reduce* to *Parameter Server* training? In this section, an empirical rule is discussed which decides online throughout the training process whether it is the right time to proceed with *Parameter Server* training. Suppose $v_{loss}(i)$ is the validation loss in epoch i and k is the last training epoch finished, then when the boolean expression in 4.1 becomes true, the training continues under an asynchronous *Parameter Server* setup.

$$s = \sum_{i=0}^{4} \frac{\|v_{loss}(k-i) - v_{loss}(k-i-1)\| \cdot 100\%}{5 \cdot v_{loss}(k-i-1)} < 1\% \tag{4.1}$$

The idea behind this empirical rule lies on changing to asynchronous training when the training process has become more stable, *i.e.,* the loss does not present very radical changes. In order to measure how stable the training is at a specific epoch, a 5-window running average over the percentage change of the validation loss is used (value s in 4.1). When the mean percentage change becomes small and

less than 1%, this epoch is considered a good switching point. Both the window size and the percentage threshold were found empirically. Larger window sizes were found sensitive to previous epochs with larger values of validation loss, leading the training to switch to asynchronous mode at the very last few epochs. Smaller window sizes might lead to earlier switching points, affected by smaller loss change between less successive epochs. The choice of the 1% threshold is further explained in Section 4.4.2

Strategy-Switch enriched with the empirical rule (ER-SS) is described in Algorithm 4.8. A detailed example on how ER-SS algorithm operates is provided in example 4.1.

Example 4.1 (Strategy-Switch Decision Example) *Consider the Empirical Rule Strategy-Switch criterion given by equation 4.1.*

Assume validation losses over epochs 1 *through* 6 *are:*

Epoch k	*Validation Loss* (v_{loss})
1	*0.500*
2	*0.450*
3	*0.420*
4	*0.415*
5	*0.413*
6	*0.412*

Evaluating at epoch $k = 6$*:*

$$
\begin{aligned}
s = &\frac{|0.412 - 0.413| \times 100\%}{5 \times 0.413} + \frac{|0.413 - 0.415| \times 100\%}{5 \times 0.415} \\
&+ \frac{|0.415 - 0.420| \times 100\%}{5 \times 0.420} + \frac{|0.420 - 0.450| \times 100\%}{5 \times 0.450} \\
&+ \frac{|0.450 - 0.500| \times 100\%}{5 \times 0.500} \\
&\approx 0.0484\% + 0.0964\% + 0.2381\% + 1.3333\% + 2.0000\% \\
&= 3.7162\% > 1\%.
\end{aligned}
$$

Thus, the criterion is not met and Strategy-Switch is not activated at epoch 6, continuing the training with All-Reduce to find a better starting point for Parameter Server.

Now consider validation losses at epochs 10 *through* 15*:*

Epoch k	*Validation Loss (v_{loss})*
10	*0.4000*
11	*0.3998*
12	*0.3997*
13	*0.3996*
14	*0.3995*
15	*0.3994*

Evaluating at epoch $k = 15$*:*

$$s = 5 \times \frac{|0.3994 - 0.3995| \times 100\%}{5 \times 0.3995}$$
$$\approx 5 \times 0.0050\% = 0.0250\% < 1\%.$$

In this case, the Strategy-Switch condition is fulfilled, and thus the switch is triggered at epoch 15*, initiating asynchronous training with Parameter Server at a point where it will be less vulnerable to staleness.*

4.4 Evaluation under network/straggler scenarios

In this section, a detailed experimental evaluation on *Strategy-Switch* is provided, compared to *All-Reduce* and *Parameter Server* training on homogeneous and heterogeneous clusters. While the experiments focus on ResNet-based architectures for CIFAR datasets, the proposed approach is not restricted to ResNets. The primary reason for choosing these networks is the wealth of published results and well-understood convergence properties, making them particularly suitable for demonstrating the effects of switching strategies.

4.4.1 Experimental Setup

The clusters and benchmarks used are the same as the ones used in the benchmarking analysis in Section 4.1. For ease of reading, #B1 and #B2 benchmarks refer to training CIFAR-10 on ResNet-20 and CIFAR-100 on ResNet-32 respectively. #C1 and #C2 refer to the homogeneous and heterogeneous clusters used (see Table 4.3 in Section 4.2 for cluster structures). As global hyperparameters, the ones discussed in the official ResNet paper [13] and also mentioned in Table 4.1 are used. Every experiment has been executed 3 times and error bars outline

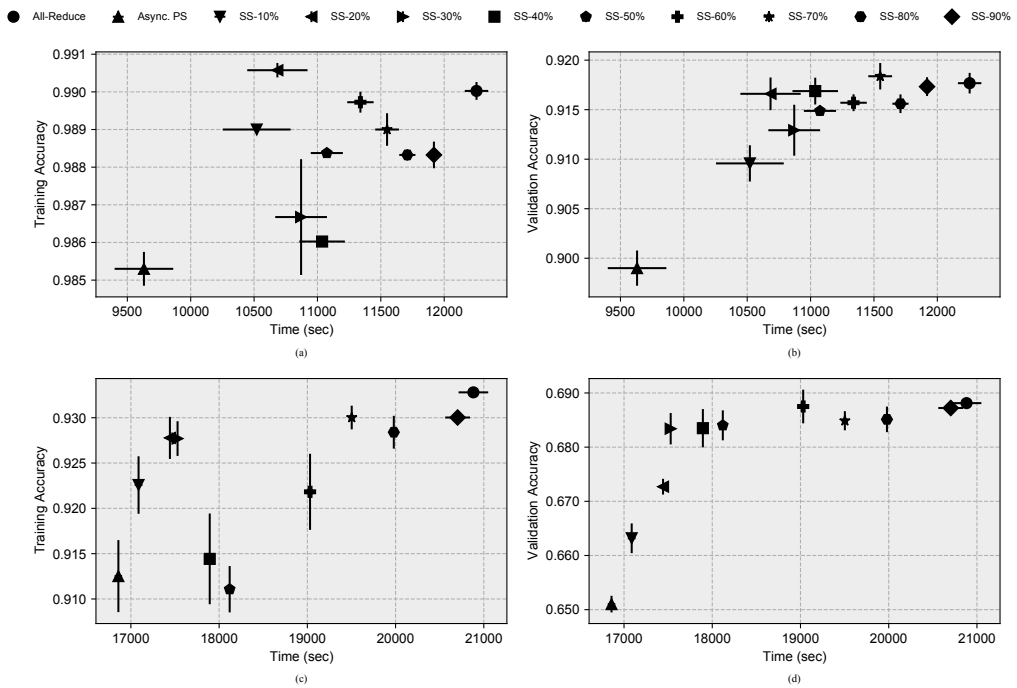


Figure 4.9: Accuracy vs. time trade-offs in #C1 cluster for each benchmark when using *All-Reduce*, *Parameter Server* and *Strategy-Switch-*α% with various switching points. Percentage in *Strategy-Switch* labels indicates the value of α%. (a) #B1 - Training Acc. (b) #B1 - Validation Acc. (c) #B2 - Training Acc. (d) #B2 - Validation Acc.

the statistical information (i.e., average and min/max values) for every experiment. The collected metrics for each experiment run were consistently close to each other, mainly due to the fact that the selection of dataset sizes and cluster resources resulted in sufficient execution times, minimizing statistical errors that may occur when the execution times are negligible. In sections 4.4.2 and 4.4.3, tradeoffs in Strategy-Switch-α% and the empirical rule based on the validations loss of the *All-Reduce* training on the benchmarks in the homogeneous cluster are discussed. In sections 4.4.4 and 4.4.5 *All-Reduce*, *Parameter Server* and Empirical Rule *Strategy-Switch* are evaluated on both the benchmarks and on both clusters.

4.4.2 Trade-offs when using *Strategy-Switch-*α% on the homogeneous #C1 cluster

Figure 4.9 presents the trade-offs regarding accuracy and execution time between *All-Reduce*, *Parameter Server* and *Strategy-Switch-*α% strategies in the homogeneous #C1 cluster.

As observed from Figure 4.9 *All-Reduce* achieves the highest train and validation accuracy for both benchmarks #B1 and #B2 in the homogeneous #C1 cluster. This can be attributed to the synchronous nature of *All-Reduce*. Therefore, the convergence of the optimization algorithm used in the training remains the same as the one of the single node. However, *All-Reduce* appears to be the slowest approach compared to the other experiments, due to the synchronization-related overheads present throughout the training process.

Unlike *All-Reduce*, *Parameter Server* achieves the lowest accuracy for both benchmarks #B1 and #B2 in the homogeneous #C1 cluster. The lack of synchronization in the *Parameter Server* harms the convergence of the optimization algorithms, due to stale gradients mentioned in Section 4.1.2. In further detail, when training is completed under the *Parameter Server*, model parameters might have been updated in various steps with gradients computed on outdated model parameters. On the other hand, asynchronous training gives an advantage to *Parameter Server*, compared to the synchronous approach in terms of the training speed. Each worker trains the model completely independently at its own highest pace leading to the fastest possible training in the *Parameter Server* for both benchmarks #B1 and #B2.

Strategy-Switch-α% proposed in this paper, achieves a balance between *All-Reduce* and *Parameter Server* by exploiting the advantages of both strategies. In *Strategy-Switch*, the training of the model starts with the *All-Reduce* and ends with the *Parameter Server* strategy. Therefore, the first α% slower epochs following the *All-Reduce* paradigm lead the model to a set of parameters closer to the optimization point, which is less prone to stale gradients of the *Parameter Server* training of the last epochs. Thus, similar levels of accuracy to *All-Reduce* can be achieved faster. α is a hyperparameter in *Strategy-Switch*, which is studied by tuning α in the range $[10, 90]$ with step 10. Irrespective of the value of α, Figure 4.9 indicates that, in *Strategy-Switch*, the accuracy metrics are higher than the *Parameter Server* and the training time is shorter compared to *All-Reduce*. Furthermore, the larger the value of α, the slower the training process is, due to more epochs performed under *All-Reduce*. In general, larger values of α indicate models that approach the convergence point of the *All-Reduce* training.

4.4.3 Explaining the s value of the empirical rule on the homogeneous #C1 cluster

As explained in Section 4.4.2, α is a hyperparameter in *Strategy-Switch*. To identify a proper switching point, the empirical rule discussed in Section 4.3.3 is proposed, leading to Empirical Rule - *Strategy-Switch*. In this section, the evolution of the s value of the empirical rule per training epoch is discussed. In Figure 4.10 the s value is presented per train epoch on the *All-Reduce* training for both benchmarks. The vertical dashed line indicates the switching point according to the empirical

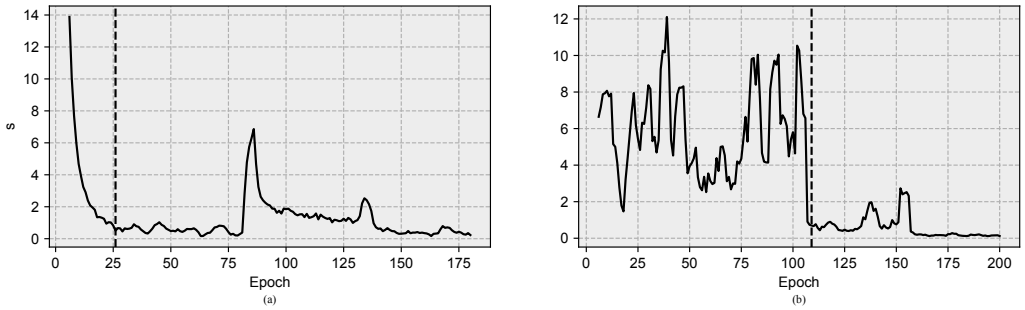


Figure 4.10: Value used in the empirical rule per training epoch on *All-Reduce* setups for both benchmarks. Vertical dashed line indicates the switching point according to the empirical rule. (a) #B1 - Validation Loss (b) #B2 - Validation Loss.

rule.

Figure 4.10a outlines the evolution of the s value for the #B1 benchmark. The validation loss drops to lower values rapidly and is stabilized early in the training process. Specifically, in epoch 26, the s value appears to satisfy the threshold of 1% in the empirical rule presented in Equation 4.1 of Section 4.3.3. For ~ 50 epochs, the value of s is at the same levels, presenting some spikes later on some epochs. Larger s values indicate larger validation loss change. Attributed to the tuning of the learning rate (Table 4.1), which is decreased at epoch 81, the validation loss presents a larger variation at this point. However, the earlier steady state of the model renders epoch 26 a good switching point, since the spikes attributed to the decrease of learning rate in the s value, are smoothed after a few epochs.

Similar observations are made in Figure 4.10b regarding benchmark #B2. However, the training appears to be stabilized at epoch 109, since benchmark #B2 is more complex to converge compared to benchmark #B1.

In the following sections 4.4.4 and 4.4.5 Empirical Rule - *Strategy-Switch* is evaluated in contrast with *All-Reduce* and *Parameter Server* for both benchmarks in the homogeneous #C1 and the heterogeneous #C2 clusters respectively.

4.4.4 *Strategy-Switch* in the homogeneous #C1 cluster using the empirical rule.

Figure 4.11, Figure 4.12 and Figure 4.13 present the results of *Strategy-Switch* in the homogeneous #C1 cluster when using the empirical rule and compares them with *All-Reduce* and *Parameter Server* training. It can be observed that in both benchmarks #B1 and #B2 *Strategy-Switch* has the best trade-off between accuracy and execution time.

When reaching convergence, training, and validation accuracy are shown in

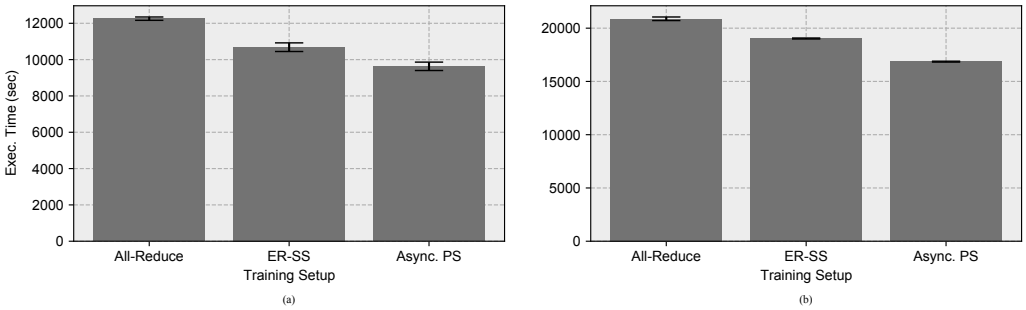


Figure 4.11: Training time under *All-Reduce*, *Strategy-Switch* and *Parameter Server* in the homogenous #C1 cluster (a) #B1 benchmark (b) #B2 benchmark

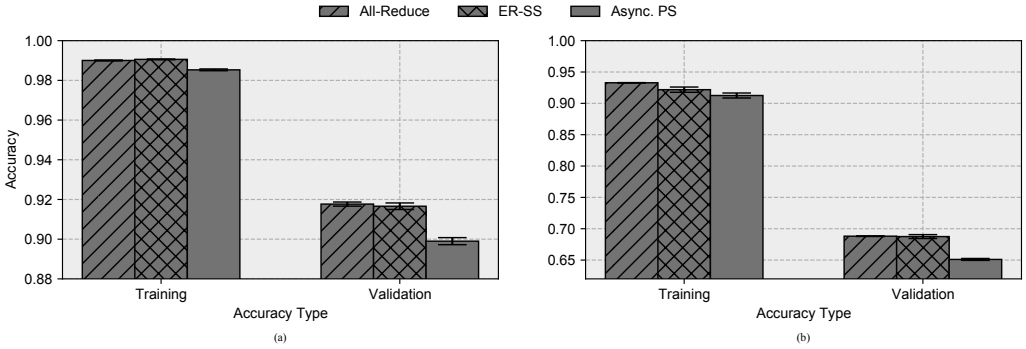


Figure 4.12: Training and validation accuracy values at convergence, when training in the homogenous #C1 cluster for the benchmarks (a) #B1 benchmark (b) #B2 benchmark

Figure 4.12a and Figure 4.12b. Training accuracy presents almost identical values between *All-Reduce* and *Strategy-Switch*, while smaller values are observed under *Parameter Server*. The same patterns are also identified regarding validation accuracy under convergence. Figure 4.11a and 4.11b present the execution time of all distributed training approaches for benchmarks #B1 and #B2 in the homogeneous #C1 cluster. *Parameter Server* strategy is the fastest one as expected. *Strategy-Switch* is clearly faster than *All-Reduce* strategy for both benchmarks #B1 and #B2. In further detail, for the #B1 benchmark, the converged *Strategy-Switch* model presents only a loss of 0.05% and 0.1% in the training and validation accuracy respectively compared to the *All-Reduce* model, while it is trained $1.14X$ faster. Regarding the #B2 benchmark, with a loss of 1% and 0.06% in the training and validation accuracy, the resulting model in *Strategy-Switch* is training with $1.1X$ speedup.

Figure 4.13c and 4.13d outline the training and validation accuracy at each

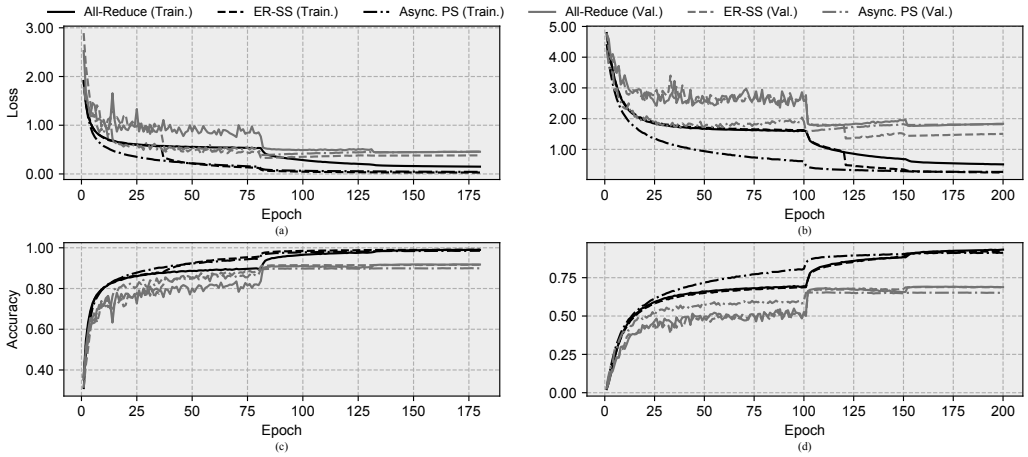


Figure 4.13: Loss and accuracy, when training in the homogenous #C1 cluster for the benchmarks. Black lines indicate training metrics and gray lines indicate validation metrics. (a) Loss - #B1 benchmark (b) Loss - #B2 benchmark (c) Accuracy - #B1 benchmark (d) Accuracy - #B2 benchmark

epoch during training the benchmarks #B1 and #B2. It can be observed that when training under *Strategy-Switch*, training loss follows the same pattern as the case of training under *All-Reduce*, while in *Parameter Server* the loss presents a different evolution. The training setups have the same behavior regarding validation accuracy, as further outlined in Figure 4.13. The evolution of the training and validation loss across epochs for benchmarks #B1 and #B2 are illustrated in Figure 4.13a and 4.13b. It is important to note that validation loss under *Strategy-Switch* evolves into smaller values across epochs compared to *All-Reduce* and *Parameter Server*. Regarding training loss, under *Strategy-Switch* similar trends to *Parameter Server* training are achieved, which evolves better compared to *All-Reduce*.

4.4.5 *Strategy-Switch* in the heterogeneous #C2 cluster using the empirical rule

Figure 4.14, Figure 4.15 and Figure 4.16 present the results of *Strategy-Switch* when using the empirical rule in the heterogeneous #C2 cluster in comparison with results from models trained under *All-Reduce* and *Parameter Server*. As in Section 4.4.4, *Strategy-Switch* presents the best trade-offs between the two baseline distributed approaches.

Figure 4.16 indicates similar trends in the evolution of loss and accuracy metrics to the ones derived from training in the homogeneous cluster. Specifically, *Strategy-Switch* models follow similar trends to the ones of the *All-Reduce* models regarding

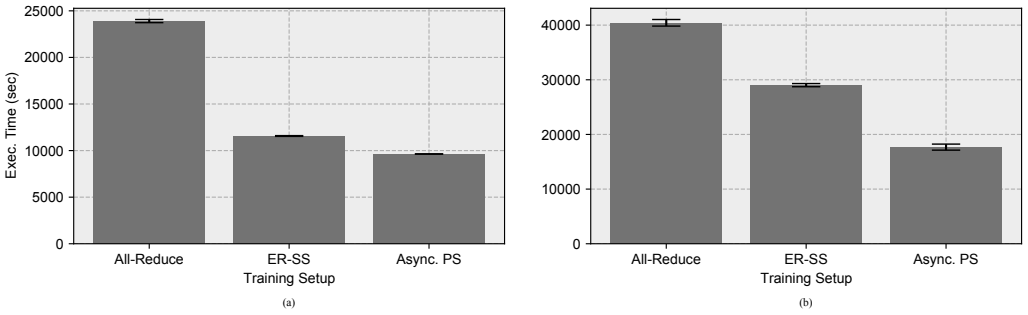


Figure 4.14: Training time under *All-Reduce*, *Strategy-Switch* and *Parameter Server* in the heterogeneous #C2 cluster (a) #B1 benchmark (b) #B2 benchmark

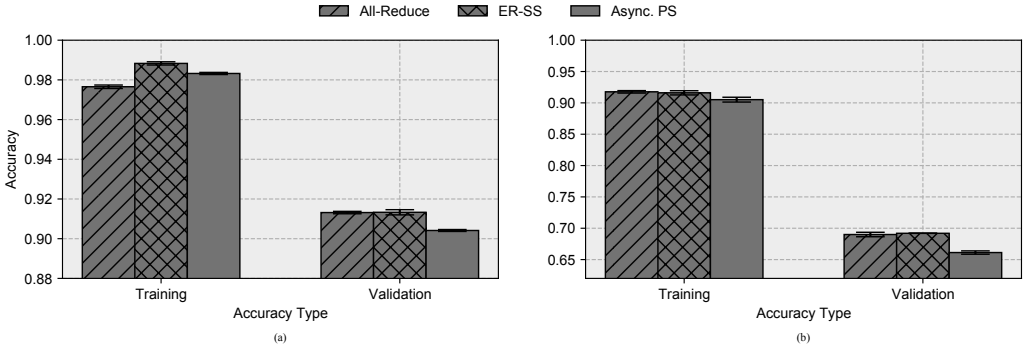


Figure 4.15: Training and validation accuracy values at convergence, when training in the heterogeneous #C2 cluster for the benchmarks (a) #B1 benchmark (b) #B2 benchmark

accuracy. In terms of training and validation loss, *Strategy-Switch* finally reaches lower levels compared to the other training setups.

The greatest difference between the results of the heterogeneous #C2 cluster and the homogeneous #C1 cluster is related to the execution time. Let us discuss in further detail the trade-offs between accuracy and training time in the heterogeneous #C2 cluster. For the benchmark #B1, Figure 4.14a and Figure 4.15a present the execution time and resulting accuracy values when training models under the three distributed approaches. *Strategy-Switch* completes the training $2.07X$ faster compared to *All-Reduce*. *Strategy-Switch* also appears to present the greatest value in both training and validation accuracy ($\sim 0.05\%$ greater than *All-Reduce*). In *Strategy-Switch* the model parameters are initialized by *All-Reduce* for the rest of the training to be performed under the *Parameter Server*. Due to the heterogeneity of the cluster and the lack of synchronization in the *Parameter Server*, the faster workers will dominate the training until they finish, leading to fewer stale param-

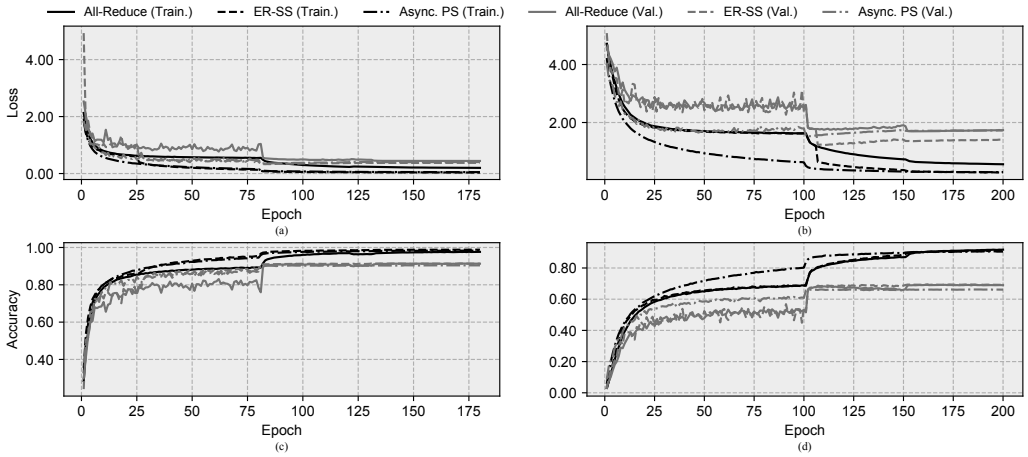


Figure 4.16: Loss and accuracy, when training in the heterogeneous #C2 cluster for the benchmarks. Black lines indicate training metrics and gray lines indicate validation metrics. (a) Loss - #B1 benchmark (b) Loss - #B2 benchmark (c) Accuracy - #B1 benchmark (d) Accuracy - #B2 benchmark

eters in the slow machines. On the contrary, the slow machines will continue the train at their own pace when the faster machines have finished, without the stale gradients effect. This observation can explain the slight increase in the accuracy values in *Strategy-Switch*. Similar trends are observed in the #B2 benchmark. In this case, execution time and accuracy values are provided in Figure 4.14b and 4.15b respectively. The model derived from *Strategy-Switch* is created $1.4X$ faster than the *All-Reduce* one, with a validation accuracy slightly enhanced by 0.19%. In both benchmarks *Parameter Server* is the faster ($2.48X$ for #B1 and 2.28 for #B2 speedup compared to *All-Reduce*), but lacks in models quality (validation accuracy harmed by 0.9% for #B1 and 2.88% for #B2 compared to *All-Reduce*).

Chapter 5

Data Distribution for the Parameter Server

As demonstrated in Chapter 3, a significant challenge associated with asynchronous training in parameter server architectures is the presence of stale gradients, which negatively impact training stability and convergence accuracy. The systematic literature review highlighted various consistency control approaches aimed at addressing this issue, primarily through synchronization techniques. Building upon these insights, this chapter explores a complementary strategy: leveraging intelligent data distribution methods to mitigate the adverse effects of gradient staleness by maintaining asynchronous training. By carefully examining how data is assigned to workers, it becomes possible to improve the consistency of updates, thus enhancing overall training performance and stability in asynchronous parameter server environments.

In this Chapter [211, 212], a vision on how the distributed deep learning process could benefit from the distribution of the training data is discussed, under the asynchronous parameter server architecture. Data preprocessing techniques can be used to obtain an a-priori knowledge of the data domain, which could be beneficial in the training process. The target is to study and propose systematic ways on how the data should be assigned to the available training worker nodes. Furthermore, random or algorithmic access patterns on data during asynchronous training will be discussed. Considering such techniques, the goal is for the training to be less sensitive to undesirable effects that appear in asynchronous distributed learning setups.

5.1 Why studying data assignment to workers?

As stated in Chapter 3, the parameter server training is usually harmed from

the stale gradients effect. Stale gradients occur when a worker computes a gradient update using old model parameters. In research works stated in the aforementioned chapter, algorithmic solutions in the parameter server or learning rate level are proposed to smooth the staleness effect. However, if the data part that can be accessed from each worker is not representative on the whole dataset, this may further harm the staleness effect, since a common approach is to randomly shard the data to workers. For instance, TensorFlow uses a modular sharding approach based on the training example index to assign it to a worker.

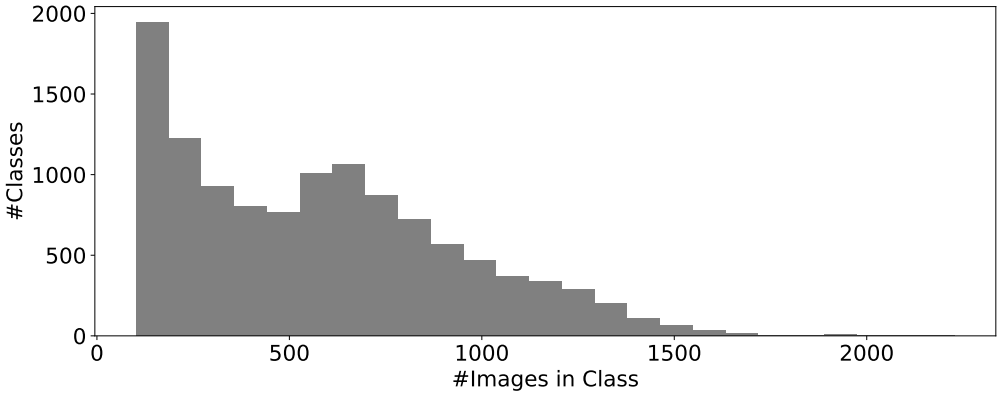


Figure 5.1: Class Population Histogram of Imagenet data obtained from Flickr.

To further support this claim, an example based on an Imagenet [14, 213] subset with images from Flickr (approx. 60GB size) will be discussed. Figure 5.1 presents a histogram with the image population in each class. In this figure, it can be observed that that most of the classes consist of approximately 100 images, while some of them consist of more than 1000 (and even more than 1500 images). Thus, it is possible that a random data assignment approach could not provide a worker with data of some of the less populated classes or bias another towards a highly populated class (data skew on some workers). Having trained on a stale parameter set on some iteration, such worker could direct the weight not towards the direction of the true optimization point, but possibly to another one which will better optimize this part of data, due to lack of knowledge regarding the data space. Part of this research aims to study whether systematic approaches in data sharding to workers could be facilitated in order to smooth the effects of staleness.

5.2 Data Modelling and Data Patterns

5.2.1 Data Modelling

Datasets that are defined as a set of characteristics with continuous values in a given space, can be defined with the help of normal distribution [214]. Normal Distribution is symmetric around a centroid mean value, while the probability of encountering a given point becomes less as the distance from the centroid grows. In a n-dimensinal space, the pdf of the normal distribution is given from the equation 5.1, where $\vec{\mu}$ is the centroid mean vector and Σ denotes the covariance table. If a vector $\vec{x}$ belongs in this normal distribution, it can be written as $\vec{x} \sim \mathcal{N}_n(\vec{\mu}, \Sigma)$.

$$f(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \cdot \|\Sigma\|}} \exp \frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu}) \tag{5.1}$$

According to definitions below, normal distribution can be used to describe datasets that are located aroung a centroid vector in the n-dimensional space and the corresponding data points are more dense closer to centroid. However, real datasets do not often comply with this structure in the multidimensional space. A usual way to describe an arbitrary dataset is using a composition of multiple normal distributions. Specifically, it can be considered that data are divided into subsets where each subset could be generated from a normal distribution. A model capable of describing this dataset modelling is named a Gaussian Mixture Model (GMM) [215, 216], which is defined in Definition 1.

Definition 1 (Gaussian Mixture Model) *A Gaussian Mixture Model (GMM) defines a distribution with a probability density function $P(x)$, as a mixture of n normal distributions with mean value vectors $\vec{\mu}_i$ and covariance tables Σ_i , $i \in \{1, 2, ..., n\}$, according to the equation*

$$P(x) = \sum_{i=1}^{n} \pi_i \cdot \mathcal{N}_i(\vec{\mu}_i, \Sigma_i)$$

where $\pi_i \geq 0$ and $\sum_{i=1}^{n} \pi_i = 1$.

Increasing the number of normal distributions used in the GMM can model more complex data structures, since the multidimensional space is divided in even smaller parts with similar data according to the generator distribution. The parameters π_i represent the probability that a specific data point comes from the i-th normal distribution of the GMM.

Suppose that the exact normal distribution that generates each data point of the dataset is known. Then the concept of a data point neighborhood can be defined, which can be considered the spherical area across the centroid of the distribution that includes all the data points from this part of the GMM. Formally, the definition of the neighborhood is provided in Definition 2. Having defined a neighbourhood, this context can be used to define when two data points are considered equivalent (Definition 3).

Definition 2 (Neighbourhood) *Let $P_0 \in \mathbb{R}^n$ a multidimensional point and $r \in \mathbb{R}^+$. The neighbourhood of the point P_0 is defined as the set of points $\{x_1, x_2, ..., x_k\}$ present in the multidimensional $\mathbb{R}^n$ space which satisfy the condition*

$$x_i \in C(P_0, r), \forall i \in \{1, 2, ..., k\}$$

where $C(P_0, r)$ represents the spherical area with P_0 as a center and radius r. The point P_0 is considered the center of the neighbourhood.

Definition 3 (Equivalent points) *Let $P, Q \in \mathbb{R}^n$ and $\mathcal{N}$ a neighbourhood in $\mathbb{R}^n$. P and Q are considered equivalent in respect to $\mathcal{N}$ if and only if*

$$P \in \mathcal{N} \text{ and } Q \in \mathcal{N}$$

5.2.2 Pattern I: Stratification

Stratification describes the way of sorting the data points into distinct groups. It is a widely used technique which is adopted in various tasks and is usually of high interests when the computing task cannot view entirely the data. For example in an approximate query processing problem [217]. Another field of interest that has expolited stratification patterns is graph partitioning [200], where the patters are exploited to isolate disjoint subgraphs.

While the aforementioned works focus on solving problems from other domains, stratification is widely used in the machine learning domain. Specifically, stratification is used to achieve more stable classification results. For instance, it is widely used in cross validation techniques for standard single node machine learning, where it reduces the metrics' variance [198]. Furthermore, in the context of learning, it is also used to facilitate lear/ning from heterogeneous databases [218].

Considering as a stratum a whole class in a classification problem, in the context of the data model described in Section 5.2.1, each stratum can represent a neighbourhood, since a spherical area that could contain all the examples from a class could always be found. Therefore, taking as an example the data assignment problem, discussed in Section 5.1, stratification could provide each training worker with an equivalent view from the data. An example is provided in Figure 5.2, where two workers participate in the training of a 4-class dataset, where the data present skew. In random assignment, there is the possibility that class 1, which contains only three training samples, might be assigned only to one worker (Worker 1), while the other worker (Worker 2) focuses more on another class, *e.g.,* class (class 4), from which Worker 1 is assigned only one sample. On the contrary, stratification ensures that both workers can study all the classes in a similar manner.

5.2.3 Pattern II: Hidden Stratification

Hidden stratification can be used to reveal how the data are organized in the distribution into non predefined subgroups. A proper example to explain hidden

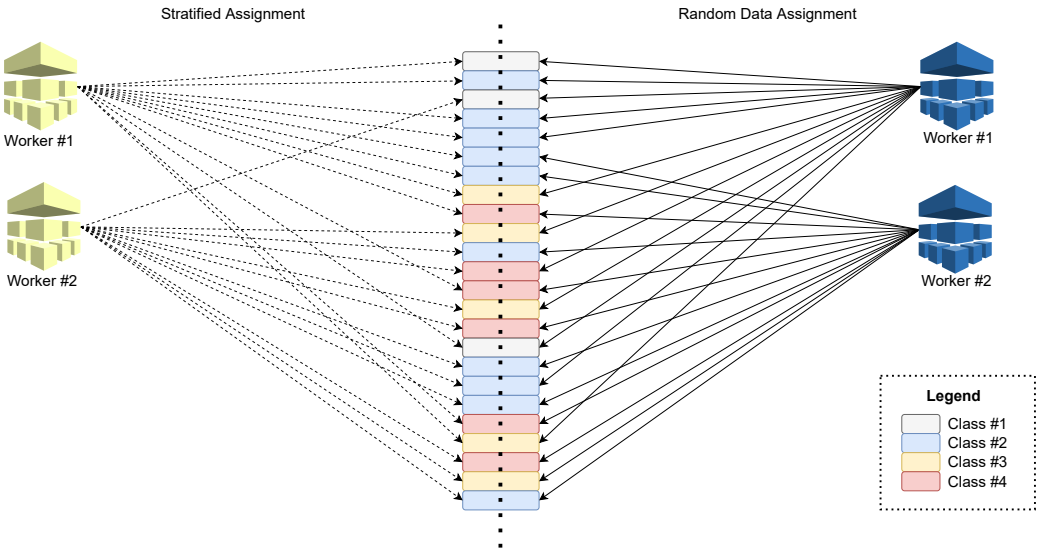


Figure 5.2: Random (right) versus Stratified (left) Assignment. Workers have access to all classes in stratified assignment, while in random each worker has a different view on the dataset.

stratification is the CIFAR-100 [206] image classification benchmark, which comes with a set of both coarse-grain and fine-grain labels.

CIFAR-100 is an image classification dataset, which comes with both a coarse-grained and a fine-grained class label for image. Figure 5.3 presents some of the coarse-grained labels of the datasets, which are analyzed in the fine-grained labels. The tree structure in the figure presents training examples from each fine-grained label. Imagine the dataset is available only with the coarse-grained labels. According to the Figure, multiple different images are characterized in the same manner, *e.g.,* both beavers and dolphins are considered aquatic mammals. However, since they are totally different animals, images from dolphins and beavers are expected to have pixel values, which come from different distributions representing different colors and formed shapes. Therefore, in case hidden stratification is explored, the fine-grained labels and the corresponding distributions could be discovered.

According to various research works, hidden stratification plays a crucial role in many machine learningrelated problems. For example, in a 2020 research [219], hidden stratification appeared to crucially affect the quality of classification models for medical images. In [220], the authors propose to explore, identify and use fine-grained labels in classification problems where only coarse-grained labels are available, achieving greater accuracy in the resulting model.

A common approach to discover hidden patterns in the data distribution is the use of unsupervised learning techniques, as clustering. Since data can be considered vectors from a multidimensional space, that are generated from a GMM, clustering can divide the space into subgroups (neighborhoods in the context of Section 5.2.1),

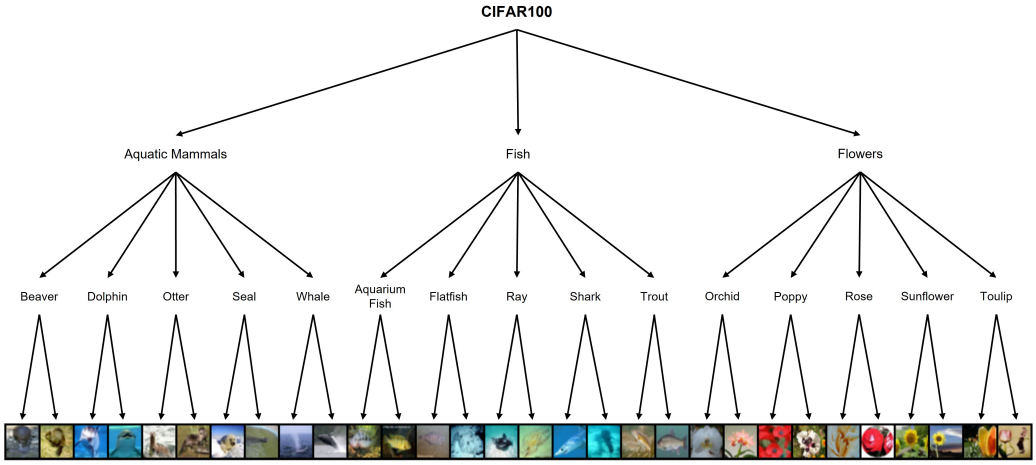


Figure 5.3: Three of CIFAR-100 coarse-grained labels split into the corresponding fine-grained labels. For each fine-grained label two example images are provided.

where it can be considered that each one is generated from one of the GMM's normal distributions.

In the big data context, multiple clustering techniques have been proposed. For instance, in [221] they have designed a clustering framework for big data, which is able to discover multiple distribution types. Others propose approximate and distributed versions of common clustering algorithms, as KMeans [222], DBSCAN [223] etc.

Apart from unsupervised learning, it would also be efficient to utilize functions that cluster together equivalent points, in the same manner as hash functions do. Towards this approach and in the spirit of Locality Sensitive Hashing, Gao proposed in [201] Data Sensitive Hashing (DSH), where he facilitates data distribution to hash together close data points in a high dimensional space. DSH is a special case of the Locality Sensitive Hashing. More information is provided in Appendix B.

5.3 Mini-batch selection: From single-node training to asynchronous distributed training

Mini-Batch SGD does not move the weights of the neural network directly to the minimization point due to the restricted view it has on the data on each iteration. However, it is known that Gradient Descent is able to move directly towards the optimization point. A usual approach to overcome such problems when training deep learning models is to randomly shuffle the data before each mini-batch extraction in order to obtain a mini-batch with less correlated data [9].

Therefore, it is reasonable to state the question what will happen if the mini-batch is chosen such that it is actually representative of the whole dataset. Will this either result to a more accurate model or to a faster training process in respect to the random sampling techniques used? Is it important to perform some preprocess to the training data in order to understand their structure and determine the sampling process during the mini-batch selection?

In 2009, Bengio proposed Curriculum Learning [224] as an approach towards this direction and proved that the training was able to converge to better local minima when he decided to use traits of the data to help the network training process. For instance, in an image classification case of categorizing shapes into elliptical, triangular and rectangular, he decided to use only some easily distinguished data at first, and then include more complex images.

Researchers from Germany have also proposed in 2016 the idea [225] of online batch selection in the training process. In this case, each training example is modelled with a probability to be selected according to its loss and the recency of its last selection. This idea enhanced the loss and accuracy on the MNIST [226] dataset. In this same spirit in 2019, the authors in [227] propose a submodular mini-batch selection method, where the model various scores that lead to which data points are selected for each mini-batch. Latest works [228, 229] attempt mini-batch selection using sliding windows on the correct predictions per sample. Depending on how accurate the model is on predicting each sample, the corresponding data point is preferred to be selected on the next mini-batch. Since mini-batch selection can boost the learning process, this technique will be able to smooth the effects of staleness in asynchronous learning setups.

Chapter 6

Offline Data Sharding for Stabilizing Parameter Server Training

In Chapter 5, the potential of systematic data distribution to mitigate the negative effects of gradient staleness in asynchronous parameter server training was identified. Building upon these conceptual foundations, this chapter delves into practical data sharding strategies aimed explicitly at stabilizing the training process. Specific offline data-sharding approaches are proposed and rigorously evaluated, contrasting them with traditional random methods (Section 6.1). By strategically organizing data before training commences, the aim is to further reduce gradient variability, improve training stability, and ultimately achieve more consistent convergence outcomes in parameter server architectures.

As a motivation experiment, a simple CNN network over the CIFAR-10 dataset is trained multiple times, having randomly distributed the data to the workers before each run. Figure 6.1 describes the architecture of the simple CNN network used. The network consists of three sequences of 2D convolutional, 2D Max Pooling, batch normalization and dropout layers and concludes some dense and dropout layers. Figure 6.2 presents the training and the validation loss per experiment. In this figure, the existence of variance between the loss values from the subsequent runs can be noticed. This observation further motivated this research over whether various sharding techniques lead to more stable results. Note that when machine learning is applied in domains, as health applications, it is important that the resulting model presents small variance in the classification accuracy. For instance, in the case of predicting the risk of diabetes, reliability model metrics depend on the standard deviation, and therefore the variance, of the model's accuracy [230]. Thus, to make more reliable models under asynchronous learning, it is crucial to further reduce the variance in the resulting metrics.

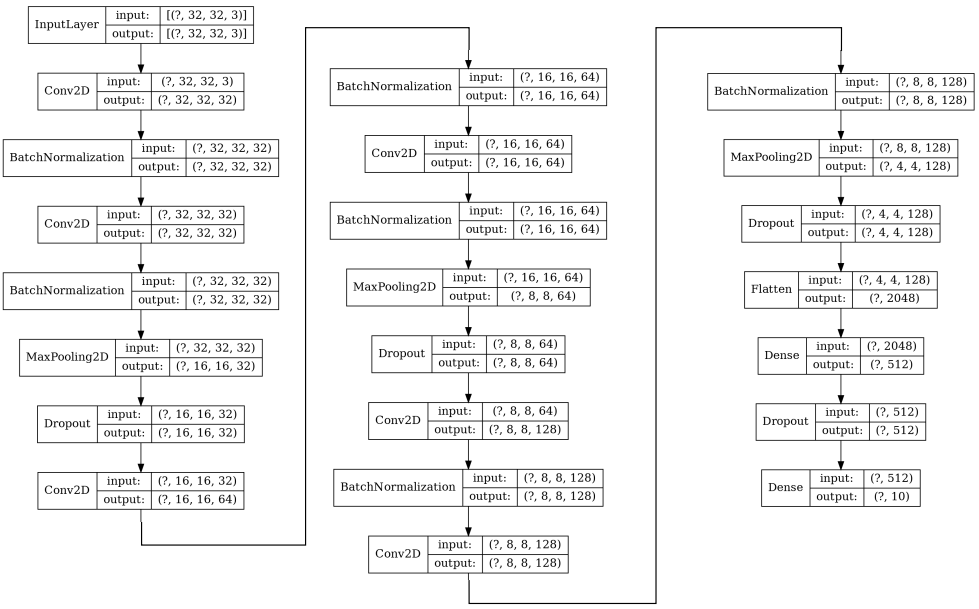
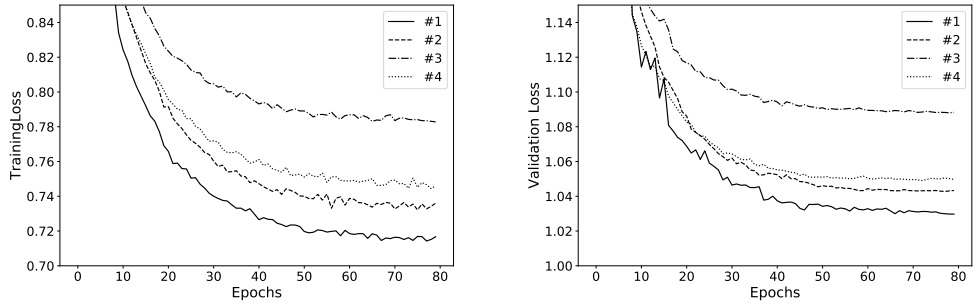


Figure 6.1: CNN Architecture



(a) Training Loss (b) Validation Loss

Figure 6.2: Loss of multiple distributed training runs of a simple CNN network over CIFAR-10 dataset.

The main contributions of this Chapter are the following:

- Two systematic sharding approaches are proposed, the Stratified and the Distribution Aware approach.
- A detailed experimental evaluation on how and why systematic data sharding can stabilize the asynchronous learning process is provided. When stratification is considered, variance of validation metrics can be reduced by up to $6X$. Distribution Aware sharding can enhance training and validation metrics in most cases with up to $8X$ and $2X$ less variance respectively.

6.1 Data Sharding Techniques

In distributed learning, under the data parallelism approach, workers contribute to the global model with gradients computed using different subsets of the data, as it was decribed in Section 2.1.2. Currently, distributed deep learning systems randomly assign data to workers. TensorFlow, for instance, uses the *shard* mechanism in its data pipeline [231], which assigns training examples to workers in a round-robin fashion. Since the purpose is to compare the effect of data assignment techniques to workers, a custom mechanism that creates the shards offline according to the method the user prefers is created. Note that it is safe to create the shards beforehand, since the equivalent mechanism of TensorFlow described, suggests to be applied at first. Thus, the data assignment is static upon the training process.

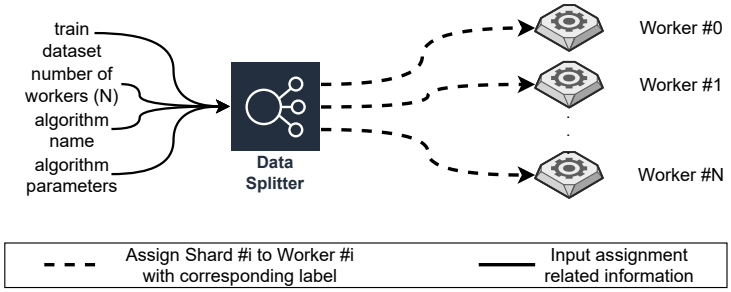


Figure 6.3: Offline Data Splitter

In this chapter, propose an offline data assignment system is proposed, outlined in Figure 6.3. This system takes as input the data set, located in a shared or distributed file system, from which the various shards shall be created, the number of workers (N) that participate in the training process and the algorithm that will assign training examples to workers followed by any related parameters. Having provided this information, the related algorithm is used to create N shards from the dataset, by assigning worker indicative labels to each training example.

In further detail, three different algorithms regarding the data shard creation process are examined. As a baseline, a random data assignment to workers is used. Furthermore, the two other techniques proposed are discussed in subsections 6.1.1 and 6.1.2 respectively, where the intuition behind them is also discussed.

6.1.1 Stratified Sharding

Random data assignment cannot ensure that each worker will have an equivalent view on the train data set. For instance, let us present a scenario where only two classes are present in the train data set, where each class represents half of the available data. Depending on the order of the training examples, some workers

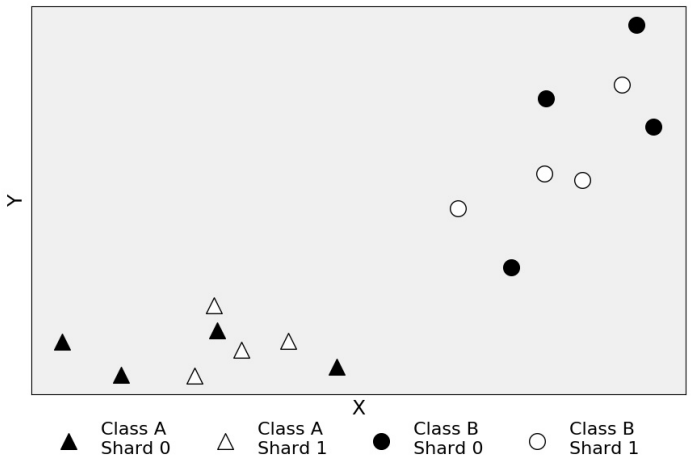


Figure 6.4: Stratified assignment: Half of the points from each class are assigned to each shard.

may well have available more examples from the one class compared to those from the other. Thus, the workers will not have a representative view of the world that these data describe, but each one will recognize one class as dominant regarding its frequency. In the aforementioned scenario, workers may attempt to lead the parameters of the global model to a different direction from each other, such that the model will more efficiently categorize the examples available in their local shard, affecting the efficiency of the resulting model.

Inspired from machine learning basics, discussed in section 5.2, stratified assignment is proposed as another technique, which is able to guarantee that each worker will be provided with a data shard that is equivalent to the world the whole train set describes. The aforementioned equivalence refers to each shard consisting of the same percentage from each class compared to the one on the whole train data set. The result of creating shards using the stratified approach is illustrated in Figure 6.4, where 2-dimensional data forming two classes are split into two distinguished shards.

Figure 6.5 outlines the algorithm used to perform a stratified assignment. The first step is to find the set of distinct classes, given the array of labels (line 5). Sequentially, the set of training examples from each class is retrieved (line 7). Finally, for every such set the data are given in a round robin manner to the workers (lines 9 - 13). This approach guarantees that each class percentage remains intact in the sample compared to the one on the whole train set.

6.1.2 Distribution Aware Sharding

Stratified assignment is an obvious way to provide the participating workers with an equivalent view of the data. However, apart from the obvious class strat-

```
1: procedure StratifiedSharding(x, y, n)
2:                                  ▷ x is an array of multidimensional tensor representing
                                      training examples
3:                                  ▷ y is the correspongin array of labels
4:                                  ▷ workers have identifiers 0, 1, ..., n-1
5:     classes = unique(y)                                  ▷ Identify available classes
6:     for each class in classes do
7:         x_class, y_class = in_class(x, y, class)
8:         class_size = length(y_class)
9:         for i = 0 to class_size − 1 do
10:                                          ▷ Round Robin split for each class
11:            worker_id = i mod n
12:            Assign example x_class(i) to worker_id
13:            Assign label y_class(i) to worker_id
14:        end for
15:    end for
16: end procedure
```

Figure 6.5: Stratified data sharding to workers using mod

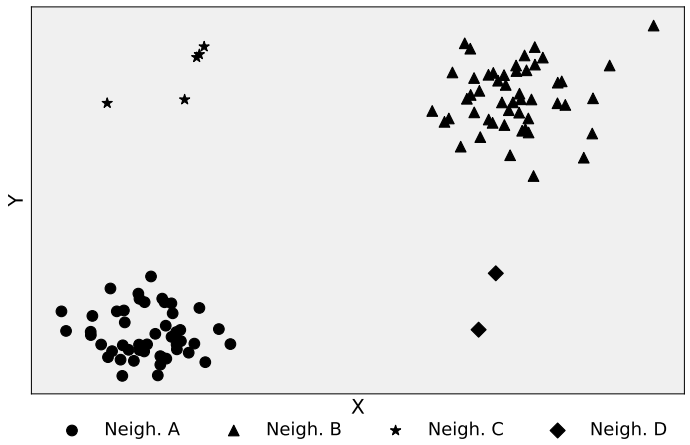


Figure 6.6: Example of 2-dimensional points organized in dense (A, B) and sparse (C, D) neighbourhoods.

ification, hidden stratification may also exist in the data set. Data can be further categorized to neighbourhoods (clusters), based on their position in respect to other data points.

Distribution aware sharding takes into account the neighbourhoods, in the context of Definition 2 given in Section 5.2.1, in the same manner the stratified one uses the classes. Before this type of assignment is described, an example of neighbourhoods from the 2-dimensional space is provided (Figure 6.6). While some neighbourhoods (A, B) consist of adequate equivalent points, others (C, D) consist

```
1: procedure DistrAwareSharding(x, y, n, classes)
2:                          ▷ x is an array of multidimensional tensor representing
                              training examples
3:                          ▷ y is the correspongin array of labels
4:                          ▷ workers have identifiers 0, 1, ..., n-1
5:                          ▷ classes is the number of classes for the KMeans algo-
                              rithm
6:     x_flattened = flatten(x)                    ▷ Flatten each example
7:     x_distribution = PCA(x_flattened)
8:     cluster_ids = KMeans(x_distribution, classes)
9:     clusters = unique(cluster_ids)                    ▷ Identify clusters
10:    for each cluster in clusters do
11:        x_cluster, y_cluster = in_cluster(x, y, cluster)
12:        cluster_size = length(y_cluster)
13:        if cluster_size > n then
14:            for i = 0 to cluster_size − 1 do
15:                                          ▷ Round Robin split for each cluster
16:                worker_id = i mod n
17:                Assign example x_cluster(i) to worker_id
18:                Assign label y_cluster(i) to worker_id
19:            end for
20:        else
21:            Assign each x(i) in x_cluster to all workers
22:            Assign each y(i) in y_cluster to all workers
23:        end if
24:    end for
25: end procedure
```

Figure 6.7: Distribution aware data sharding to workers using mod

of isolated examples. It is important to mention that it cannot be determined whether these isolated points are outliers or if the available data set does not consist of more equivalent points. Since such knowledge is not available, outlier points forming sparse neighbourhoods should be handled in a different way from this type of assignment. Thus, the algorithm distinguishes two different cases:

1. *Densely populated neighbourhoods:* Data points from such neighbourhood shall be assigned in a round robin fashion to workers. This assignment provides each worker an equivalent view of this neighbourhood.

2. *Sparsely populated neighbourhoods:* Each worker should be also provided with a view from such neighbourhoods. Since the data points are insufficient to split, the whole neighbourhood is broadcasted to all workers.

Distribution aware assignment is fully outlined in Figure 6.7. First, in order to have data points in $\mathbb{R}^n$, each training example is flattened (line 6). KMeans algorithm [232, 233] is used to identify the various neighbourhoods that can be formed by the available data (line 8). Note that the algorithm takes as input the number

of neighbourhoods that data are organized to. In order to speed up the KMeans algorithm, each data point is transformed beforehand to a format that will contain its important information summarized. A state of the art approach is to apply the PCA algorithm [234], which minimizes the information loss by maximzing variance (line 7). Finally, a round robin assignment is performed for each densely populated neighbourhood (lines 11-19), while sparsely populated neighbourhoods are broadcast to all workers (lines 21-22). Note that sparsely populated neighbourhoods are considered those with less points than the number of workers. This assumption is made, since otherwise each worker cannot have a distinct and equivalent view on this neighbourhood.

6.1.3 Time Complexity

Having presented the sharding approaches, it is important to identify their time complexity. Suppose a dataset of n training examples in the $d-$dimensional space is available. The baseline random assignment can be implemented in $\mathcal{O}(n)$, by choosing to assign each example to one of the workers with the less assigned data. Stratified sharding is implemented with two nested for loops, passing the data points once (see Figure 6.5). Thus, stratified sharding also is a $\mathcal{O}(n)$ algorithm. Regarding the distribution aware approach, its computational complexity is determined by applying the PCA and KMeans algorithms, which are used from the scikit-learn library in this implementation. PCA has a $\mathcal{O}(n \cdot d^2)$ [235] time complexity. KMeans, since the number of maximum iterations T is fixed to 150, becomes a $\mathcal{O}(k \cdot n)$ [236] algorithm, where k is the desired number of neighbourhoods. Therefore, the distribution aware sharding is a $\mathcal{O}(n \cdot max\{d^2, k\})$ algorithm.

6.2 Experimental Setup

The experimental evaluation is conducted on a cluster consisting of 15 virtual machines. Each virtual machine has 4 CPUs and 16GB RAM and operates with Ubuntu 16.04.6 LTS. TensorFlow in version 2.3 is used to build and train the models under the parameter server architecture, which was the latest stable one when performing the experiments. As a common distributed file system, where the workers save checkpoints and summaries Apache Hadoop [69] is deployed with 1 namenode and 14 datanodes.

Regarding the tasks that participate in the training, each one is deployed in a different VM. In further detail, 2 servers, 12 workers and 1 evaluator task are deployed. In the training process, each worker exploits only data assigned to them from the sharding algorithm.

Note that there are no GPUs available on this experimental cluster. However, since it is needed to evaluate how the model quality under an asynchronous training

Table 6.1: Dataset Characteristics

Dataset Name	Image Size	Train Size	Test Size	#Classes
CIFAR10	(36,36,3)	50,000	10,000	10
CIFAR100 (Coarse)	(36,36,3)	50,000	10,000	20
CIFAR100 (Fine)	(36,36,3)	50,000	10,000	100

Table 6.2: Resnet-56v1 Singlenode Training Configuration

Configuration Name	Value
Epochs / Optimizer	182 / SGD with momentum 0.9
Learning Rate	Initial 0.1, divide with 10 after 90 and 135 epochs
Normalize Dataset	Subtract Mean of Training Images

setup is affected from the algorithm used to create the data shard, the lack of such accelators is not crucial for the quality of the experiments.

6.2.1 Datasets, Networks and Training Setup

fThe proposed data sharding schemes are evaluated with benchmarks from the Image Classification domain. The experimental evaluation for the different data assignment strategies is performed on training the ResNet-56v1 network [13] with the CIFAR-10 and CIFAR-100 [206] respectively. CIFAR-100 is examined both with the coarse and fine grain labels available. More details regarding these data sets are presented in Table 6.1. Regarding the network configuration, [13] clearly proposed a set of hyperparameters for training on the CIFAR datasets in the single node case (outlined in Table 6.2). Taking the aforementioned single node training configuration into account, all the necessary hyperparameters aere adjusted for the distributed setup as explained in section 2.1.2.

For each experiment five runs are performed and the mean values and the variance of the final loss and accuracy over the training and the validation set are used. Any effects that occur by using each sharding approach are further discussed. Regarding the Distribution Aware technique, the same global data set size as the baseline is considered. In this way, the model is not further trained with more mini - batches per epoch, but study how sparse neighbourhoods can affect the training results.

Table 6.3: Statistics (5 runs) of final Training and Validation Loss / Accuracy on the CIFAR-10 dataset per method

Method			**Training Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			0.165786	0.001567	0.920000	0.001086
Stratified			0.165751	0.001312	**0.922397**	0.000736
Distribution Aware	Class Number	20	0.165237	0.002471	0.921583	0.004059
		30	0.165166	0.001270	0.922218	**0.000128**
		40	**0.165046**	**0.000761**	0.921884	0.000851

Method			**Validation Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			0.442370	0.006174	0.935489	0.000863
Stratified			0.444076	0.005401	0.935396	**0.000284**
Distribution Aware	Class Number	20	0.445127	0.004059	0.935291	0.000648
		30	0.442075	**0.003431**	**0.935885**	0.000550
		40	**0.441692**	0.011402	0.935127	0.001358

6.3 Experimental Evaluation

6.3.1 CIFAR10

Metrics and Variance.

Table 6.3 outlines the mean and variance of training and validations metrics for each sharding approach applied on the CIFAR-10 dataset. While using stratified sharding led to similar reduction of the train loss, it is important to notice its effect on the variance of the metrics. Stratified sharding concludes in training and validation accuracy with $1.47X$ and $3.03X$ less variance compared to the baseline. Loss values also follow similar patterns.

Distribution aware mechanism leads to a slight enhancement in the metrics and might further the decrease of the variance. For instance, when sharding using 30 neighbourhoods, the validation accuracy meets the best value of 0.9359. Training accuracy is enhanced by 0.22% with a $8.48X$ less variance compared to random. Compared to the baseline, variance of validation metrics is also enhanced by up to

Table 6.4: Sparsely populated neighbourhoods through Distribution Aware Algorithm (various cluster sizes as input) with the resulting validation metrics for 3 of the runs (CIFAR-10). Mean Size refer to the mean population of all those neighbourhoods.

#**Clusters**	**Run #1**			**Run #2**			**Run #3**		
	Sparse Neighb.	Mean Size	Valid. Accuracy	Sparse Neighb.	Mean Size	Valid. Accuracy	Sparse Neighb.	Mean Size	Valid. Accuracy
20	2	1	0.9352	1	1	0.9345	3	1	0.9361
30	6	2.83	0.9351	6	1.17	0.9361	4	1.25	0.9364
40	7	1.57	0.9370	8	1	0.9345	11	1.56	0.9330

$1.57X$.

Overall, in the CIFAR-10 series of experiments, the variance of the metrics, when using the baseline random approach, appears to be greater than the one from the proposed approaches. This observation is explained if the data distribution per worker is examined. This issue will be further discussed in section 6.3.2. Distribution Aware approach only presents greater variance in the validation metrics compared to the baseline if a large number of neighbourhoods compared to the number of classes in the dataset is used. This effect is discussed in the next subsection (6.3.1).

Effect of Sparsely Populated Neighbourhoods.

Another interesting insight comes from examining how the model metrics are affected when increasing the number of classes provided in the distribution aware algorithm. The training loss and its variance are constantly decreasing with the increase of the proposed number of neighbourhoods. However, according to the validation metrics of Table 6.3, if the algorithm is provided with a large number of neighbourhoods, their variance becomes larger, indicating that the model might slightly overfit. Since sparsely populated neighbourhoods are broadcasted to all workers, it is more likely that a greater number of examples will be reused from all workers while increasing the number of neighbourhoods. For example, in the case of 40 neighbourhoods, a more closer look on each distinct run asserted the above statement. During one of the runs, as shown in Table 6.4, 11 sparsely populated neighbourhoods appear to slightly harm the validation accuracy which concluded in a value of 0.9330. On the other hand, when 7 sparsely populated neighbourhoods occurred, the validation accuracy managed to reach 0.9370. Note that the validation loss on this case was 0.4329 leading to a 2% enhancement from the results of the baseline method. Overall, while the distribution aware sharding can enhance the value of the metrics, a large number of neighbourhoods should not be examined to avoid the case of multiple sparsely populated ones.

Table 6.5: Statistics (5 runs) of final Training and Validation Loss / Accuracy on the Coarse CIFAR-100 dataset per method

Method			**Training Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			0.352169	0.004761	0.811742	0.003103
Stratified			**0.346032**	0.001903	**0.814008**	0.000766
Distribution Aware	Class Number	30	0.349379	0.003350	0.812987	0.002031
		40	0.350087	**0.001738**	0.812330	**0.000535**
		60	0.347240	0.001893	0.812624	0.001138
Method			**Validation Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			1.216487	0.010065	0.805993	0.004452
Stratified			1.232794	**0.006096**	0.805841	**0.000728**
Distribution Aware	Class Number	30	1.219970	0.009967	0.806072	0.000931
		40	1.211612	0.008745	**0.806237**	0.002374
		60	**1.209313**	0.021793	0.805775	0.003009

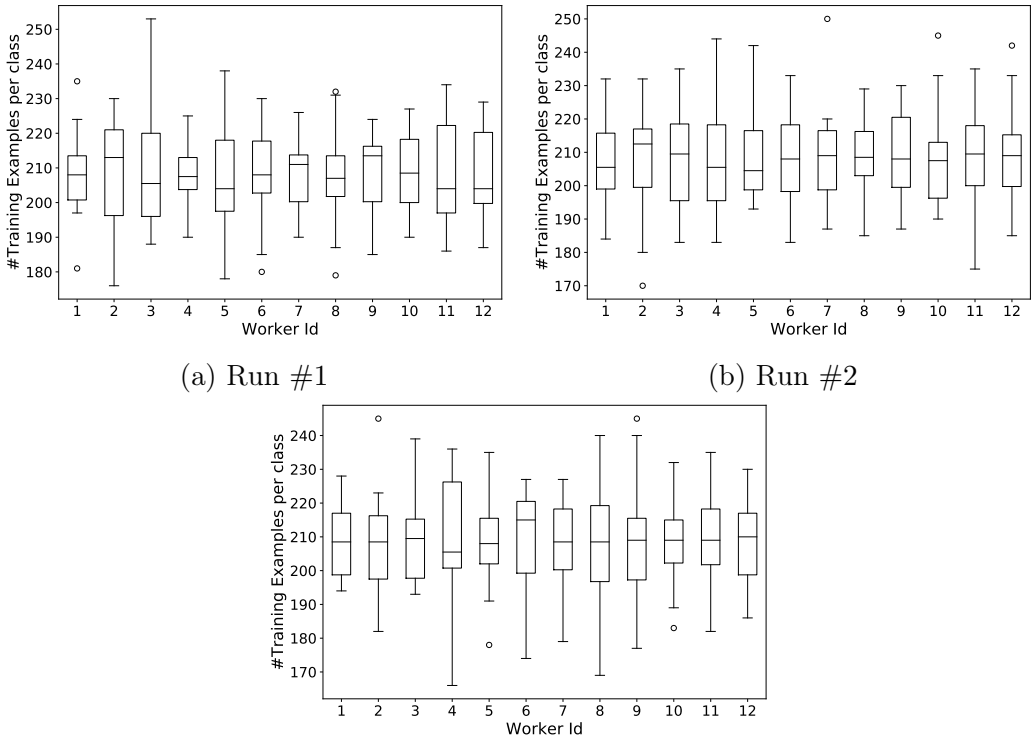
6.3.2 Coarse - Grain CIFAR-100

Metrics and Variance.

Table 6.5 presents the final training and the validation metrics of training the Resnet-56v1 network of the CIFAR-100 dataset with the coarse - grain labels, having the data sharded to workers with all the aforementioned techniques. Stratified sharding led to results that minimize the training metrics and the variance of the validation metrics. Specifically, the variance of the validation loss and accuracy was $1.65X$ and $6.11X$ smaller than the baseline. Training loss was also minimized by 2% from the one emerging from the baseline sharding.

While stratified sharding manages to minimize the variance, the actual value of the validation metrics is minimized when using the distribution aware sharding algorithm. In 6.3.1 the experimental evaluation indicated that distribution aware sharding, when given the appropriate number of clusters as input, could provide the model with the best validation metrics and smaller variance than the baseline. Table 6.5 can further support this claim. Providing the algorithm with twice the number of classes, appear to have an enhanced validation loss with $1.21X$ less variance from the round - robin sharding. The same applies for the validation

accuracy, which meets its best value of 0.806237 in this setup. Note that validation loss is minimized when the number of clusters in the algorithm is set to three times the number of classes (60). However, the variance in this setup appears to suffer from the sparse neighbourhoods effect discussed in 6.3.1.



(a) Run #1

(b) Run #2

(c) Run #3

Figure 6.8: Box plots representing the number of class examples in each shard created randomly for 3 of the runs.

Random Sharding Weakness.

Having discussed the variance minimization that can be achieved from the stratified sharding algorithm, it is important to further understand why the default random approach results to larger variance in the validation metrics. Figure 6.8 presents a group of box plots for 3 of the runs of the random sharding approach. Each box plot describes the number of training examples from each class that is assigned to each worker. CIFAR-100 consists of 20 equally populated coarse grain labels. Sharding the data set into 12 workers should provide each with approximately 208 training examples from each category. Most workers have a median number of examples per class close to this value and overall $(200, 215)$ as a 50% confidence interval in most cases. However, the box plots indicate that several

Table 6.6: Statistics (5 runs) of final Training and Validation Loss / Accuracy on the Fine CIFAR-100 dataset per method

Method			**Training Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			0.512164	0.009550	0.744448	0.007855
Stratified			0.516125	0.009296	0.744203	0.003107
Distribution Aware	Class Number	50	0.513846	**0.004085**	0.746785	**0.000764**
		100	0.521913	0.008164	0.743808	0.005842
		200	**0.510340**	0.006247	**0.746949**	0.001134

Method			**Validation Metrics**			
			Loss		Accuracy	
			Mean	Variance	Mean	Variance
Random			1.841267	0.026051	0.704707	0.002222
Stratified			**1.817435**	**0.011633**	**0.708432**	**0.001225**
Distribution Aware	Class Number	50	1.835961	0.019680	0.707048	0.004526
		100	1.851682	0.015945	0.703488	0.003561
		200	1.839176	0.013421	0.708234	0.002264

classes are distributed unequally to the workers. For instance, a closer look on the box plots referring to the second and third runs (Figures 6.8b and 6.8c) indicates that most of the workers have $180 - 230$ and $175 - 235$ training examples from each category respectively, leading to unbalanced sharding for some classes. Thus, a worker will try to adjust the model more on one specific class leading to greater variance in both training and validation metrics between the training attempts. Of course, an outlier number of examples per class could further hurt the variance, as for instance in the case of workers 2 and 7 from the second run (Figure 6.8b) and worker 2 from the third run (Figure 6.8c), since the divergence of the class size compared to the rest will further create dominant or subdominant classes.

This non - uniform view each worker has on the data, is not met on the proposed algorithms. Stratfied shards preserve the percentage of each class size from the whole dataset. Distribution aware technique, considers further hidden stratification, which also avoids this problem, if the appropriate number of clusters is given as input (Section 6.3.1).

6.3.3 Fine - Grain CIFAR100

Metrics and Variance

Table 6.6 reveals some interesting findings regarding the validation metrics. To begin with, as discussed in 6.3.1 and 6.3.2, the variance of the validation metrics is minimized when using stratified shards. Specifically, both validation loss and accuracy are less variant by $2.23X$ and $1.81X$ respectively. Apart from the variance, this series of experiments shows shards derived from the stratified algorithm, also manage to slightly enhance the values of the validation loss and accuracy to $1.81X$ and 70.84% respectively.

As it is shown in Table 6.6 training loss is not minimized by shards created from the stratified approach. When distribution aware shards are used, the model appears to minimize the training loss and maximize the training accuracy, followed by similar validation accuracy to the one of the stratified case. While the distribution aware case does not minimize the variance of the validation metrics, it also presents lower values of variance compared to the baseline method.

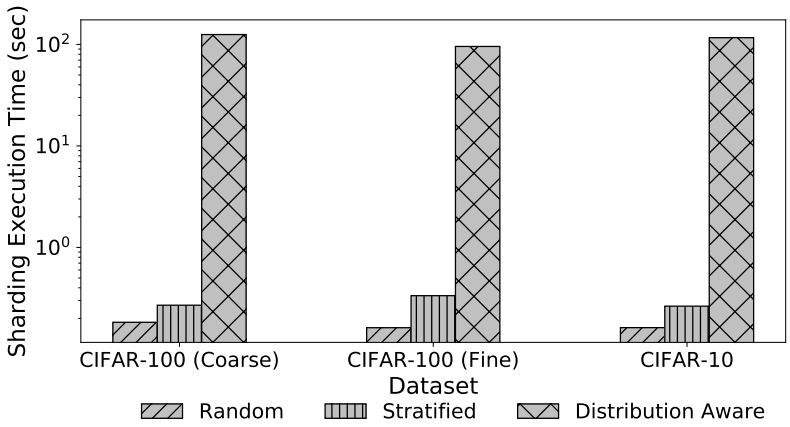


Figure 6.9: Average Sharding Technique Time per Dataset

6.3.4 Time Ovehead of Data Sharding

Figure 6.9 presents the average execution time in seconds for creating shards with technique. For the distribution aware technique the mean execution time over all classes examined for each dataset is presented. Stratified and Mod sharding techniques, as same complexity algorithms (see Section 6.1.3), induce almost the same time overhead to the whole training process, which is less than 1 sec. Distribution aware's overhead is ~ 100 seconds, which does not burden the whole training process, since the ResNet-50 network needed approximately 4 hours to converge in the cluster.

Since GPUs are not used to train the models, it shall be ensured that the time overhead induced from the sharding algorithms is not important in case accelerators are available. Let us take as an example the family of CIFAR datasets. As mentioned in 6.2.1, each of these datasets is used to train a model following the ResNet-50 architecture with the proposed hyperparameters, i.e a global mini-batch size of 128 training examples. Thus, according to 2.1.2, WB is approximately 10 training examples. Benchmarking indicated that TensorFlow needs approximately 0.11 seconds to train a ResNet-56v1 network with a WB mini-batch on a Tesla GPU [237]. In such case, each worker will need 7800 seconds and, therefore, the overhead of the distribution aware approach is insignificant.

6.4 Discussion and Conclusions

Having presented a detailed evaluation on sharding algorithms, in this section the findings are discussed to help the reader understand the benefits and the drawbacks of each one. Random sharding approach appears to have large variance in the training and validation metrics in general. As discussed in 6.3.2, workers appear to be biased towards one or more classes, which will affect the resulting global model. Stratified sharding comes as a solution to this problem, since each worker has a uniform distribution over the population of each class and, therefore, an equivalent view to the data.

Distribution Aware approach appears to be further useful when the train set presents further hidden stratification (coarse CIFAR-100). It is crucial to set the correct number of neighbourhoods in the algorithm, in order to avoid the sparse neighbourhoods effect (see Section 6.3.1). Considering the results obtained from all the CIFAR family datasets that were examined, it is recommended to set the number of neighbourhoods twice the number of classes. However, if prior knowledge indicates no hidden stratification patterns, stratified sharding should be preferred.

To generalize the results, it is important to observe the structure of each dataset. CIFAR-10 contains a few classes (10) with multiple points each (5000). On the contrary, fine-grain CIFAR-100 has multiple classes (100) which are less populated (500 data points / class). Coarse-grain CIFAR-100 is an example of a dataset with hidden patters. These different structures encapsulated by the CIFAR family datasets could allow us to generalize the findings, regarding how each sharding technique affects the variance in the metrics.

In case of applying the Distribution Aware algorithm in much larger datasets, distributed PCA and KMeans should be preferred to minimize the time overhead [222, 238].

Chapter 7

Conclusions and Future Directions

In this chapter, the contributions of this thesis are summarized and possible directions on how the data distribution could be further exploited to enhance less consistent parameter server approaches are discussed.

7.1 Conclusions

7.1.1 Revisiting the Research Question

This thesis set out to answer a single overarching question: *How far can we push the efficiency and stability of large – scale distributed asynchronous deep-learning in the parameter server architecture?*

To answer this research question, this thesis has contributed in the following direcitons:

- understanding issues related to the architecture
- adapting training consistency control levels on the fly
- letting the *data itself guide scheduling decisions*

Specifically, this thesis initially contrasted general-purpose distributed training architectures with the Parameter Server to showcase the power of this specialized training architecture. Moreover, it dissected the Parameter Server (**PS**) design space, and finally proposed two complementary techniques—**Strategy-Switch** and **data-aware sharding**—that together shrink time-to-accuracy while stabilising convergence.

7.1.2 Summary of Contributions

1. **Large-scale benchmark of MapReduce vs. Parameter Server** (Chapters 2 and **??**). On a 140-node synthetic workload an **8.23× speed-up** was measured in wall-clock time and markedly higher hardware utilization for TensorFlow's Parameter Server over Spark MLlib, pinpointing scheduler overheads and extra I/O as the root causes.

2. **Holistic survey of PS optimizations** (Ch. 3). The thesis classified more than 110 papers into five domains—*Consistency Control, Network Optimization, Parameter Management, Straggler Mitigation, Fault Tolerance*—providing a unified map that guided the subsequent design chapters.

3. **Strategy-Switch: adaptive consistency controller** (Chapter 4). A hybrid schedule that begins with synchronous All-Reduce, then automatically flips to asynchronous PS once loss fluctuations fall below an empirical threshold. Across CIFAR-10/ResNet benchmarks Strategy-Switch sustained *All-Reduce-level accuracy* while cutting training time by up to **2.1×** on heterogeneous clusters.

4. **Systematic data sharding for variance reduction** (Chapters 5 and 6). *Stratified* and *Distribution-Aware* sharding were introduced that respect class labels and hidden neighbourhoods, respectively. On asynchronous PS they lowered validation-metric variance by up to **8X** and **2X** without extra communication.

7.1.3 Synthesis and Key Insights

Taken together, the results reveal a coherent arc:

- The *architectural choice* (PS > MapReduce) delivers the first **order of magnitude** efficiency gain.
- **Dynamic protocol switching** then re-allocates that efficiency budget into *straggler tolerance*—Strategy-Switch finishes first even on skewed hardware.
- Finally, **data-centric placement** converts remaining runtime slack into *metric stability*, a dimension often ignored in pure speed papers.

These levels (system, protocol, data) are orthogonal and thus *composable*: future work can also mix-and-match them to suit cluster heterogeneity and domain constraints.

7.1.4 Limitations

1. **Scale realism**. Synthetic workloads mimic Imagenet-size tensors but not the resource fragmentation typical in shared clouds.

2. **Domain scope**. Experiments center on computer-vision tasks (CIFAR-10/100); NLP transformers may exhibit different communication/sparsity patterns.

3. **Offline sharding**. Both sharding schemes run as a pre-processing step; online re-sharding overheads remain unexplored, which is presented as a direction for future research.

4. **Heuristic switch criterion**. The empirical 5-epoch moving-variance rule, though robust in tests, lacks a convergence proof under non-IID data.

7.2 Future Directions

The main ideas include the following :

- Online data sharding approaches for mini-batch selection in the parameter server architecture
- A Data-Aware Parameter Server

7.2.1 Explore Online Sharding Techniques for mini-batch selection.

In chapter 6, two offline data sharding techniques that aim to stabilize the learning process were proposed. However, it is more promising, as discussed back in Section 5.3, to adapt data selection according to the training. To this end, information could be exploited from:

- the selected minibatch
- the model parameter it affects

The idea to exploit the training batch and the model parameters it affects, could open the way to creating DSH functions useful for the ML case. Such functions could be used to boost training acoordingly, in case the model presents inconsistencies in some classes.

7.2.2 Vision on Data Aware Parameter Server

The ultimate goal of this research is to propose a data-aware parameter server for more efficient asynchronous or stale synchronous training, adopting data related information in the mini-batch selection process as described in Section 5.3. The proposed architecture will take into account the neighborhoods, according to the definition provided in Section 5.2.1, and the training and validation metrics to create the next mini-batch. Therefore, each worker will focus on the neighborhoods

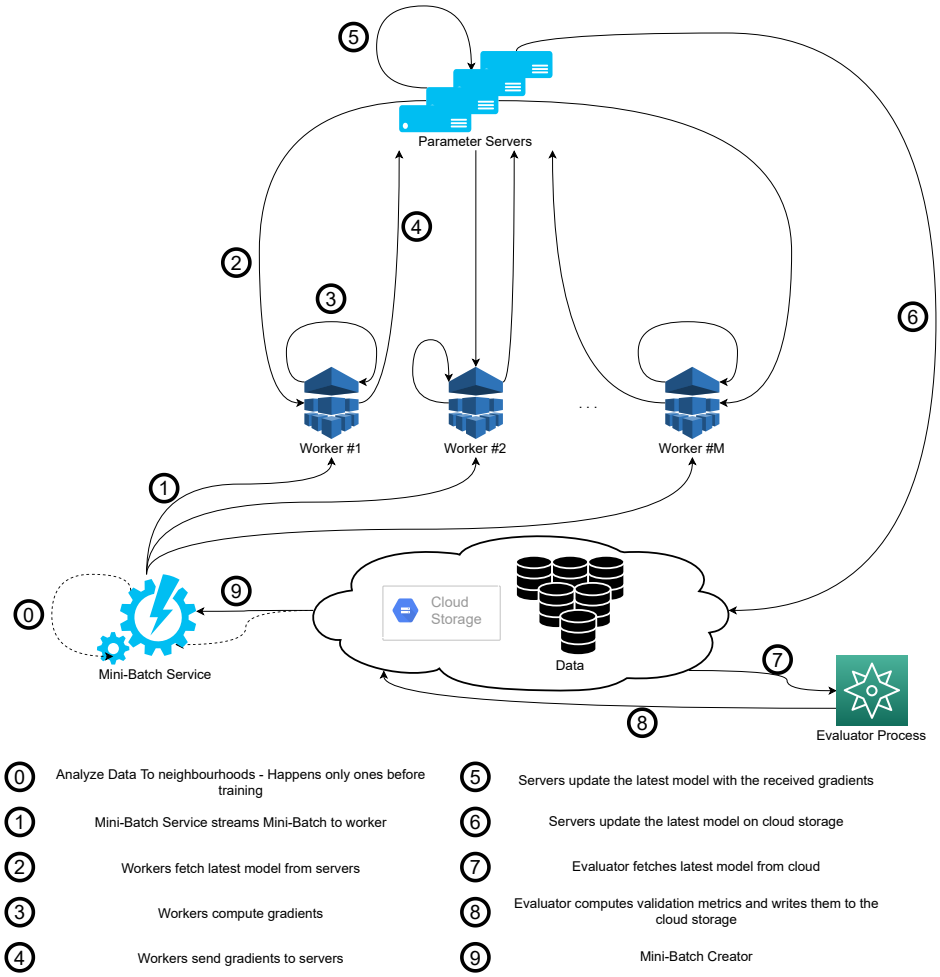


Figure 7.1: Vision of Data Aware Parameter Server Approach.

in which the model is vulnerable. The combination of these approaches will actually smooth out the effects attributed to staleness.

Figure 7.1 illustrates the data-aware parameter server that is described.

7.3 General Research Directions Based on the Thesis Survey

Chapter 3 has performed an extensive systematic literature review on the *Parameter Server* architecture by systematically categorizing and analyzing research advancements across five critical areas: consistency control, network optimization, parameter management, straggler handling, and fault tolerance. By synthesizing insights from a diverse range of studies, the trade-offs and practical effectiveness

of various optimization techniques have been outlined, providing researchers and practitioners with a foundational understanding of this domain. Looking ahead, several promising research directions emerge:

Scalability: Addressing the scalability of parameter servers in increasingly heterogeneous and large-scale environments is a critical area of ongoing research. Techniques that adapt to dynamic workloads and resource heterogeneity are becoming ever more significant. Recent advancements exemplify this trend. For instance, Megatron-LM [239] integrates inter-node pipeline parallelism, intra-node tensor parallelism, and data parallelism to optimize training across a cluster of approximately 3,000 GPUs. This approach achieves a 10% improvement in throughput, highlighting the potential of hybrid parallelism techniques. Building on these foundations, MegaScale [240] further pushes scalability boundaries by training large language models (LLMs) on a cluster of 10,000 GPUs. By improving the efficiency of Megatron-LM by a factor of 1.34X, MegaScale achieves a Model FLOP Utilization (MFU) of approximately 55%. This demonstrates that efficient technical approaches can significantly enhance scalability and resource utilization, suggesting that continued innovation in this direction is both feasible and necessary. Additionally, Microsoft's DeepSpeed team continues to extend the scalability of distributed learning frameworks. Their recent solution, ZeRO++ [110], introduces novel optimizations that improve performance and scalability in large-scale distributed training setups. These advancements underscore the ongoing effort to develop more efficient and scalable approaches, paving the way for future breakthroughs in parameter server architectures.

Standardized benchmarking: Finally, the standardization of benchmarking frameworks for evaluating parameter server optimizations is crucial to gaining clearer insights into their performance under diverse conditions. Standardized benchmarks foster meaningful comparisons and drive progress in this field. Recent findings from Stanford's AI Index Report [22] highlight a saturation of legacy AI benchmarks such as ImageNet [213], SQuAD [241], and SuperGLUE [242]. Modern models often achieve near-perfect scores on these benchmarks, rendering them less effective for evaluating current state-of-the-art systems. Notably, between 2023 and 2024, approximately 15 benchmarks were deprecated for this reason (Figure 2.1.17 in [22]). To address these challenges, new benchmarks have emerged to provide more comprehensive and specialized evaluation criteria. Generic large language model (LLM) benchmarks, such as Stanford's HELM [243] and HEMM [244], have been introduced to assess models across broader and more diverse domains. For language understanding, specialized benchmarks like MMLU [245] are gaining traction. Crowdsourced benchmarks, where human participants vote for their preferred models, represent another innovative approach, with Chatbot Arena [246] serving as a prominent example. Truthfulness evaluation is an increasingly active area of research, with benchmarks like TruthfulQA [247] and HaluEval [248] assessing models' accuracy in generating factually correct information. Reasoning benchmarks have also gained attention, exemplified by MMMU [249] and BigToM [250], which

focus on evaluating models' logical and inferential capabilities. For models functioning as agents that interact with and perform actions in dynamic environments, specialized benchmarks have been proposed. AgentBench [251] is a representative example, offering tools to evaluate models' performance in action-based scenarios. These advancements in benchmarking not only provide robust tools for assessing the performance and applicability of AI models but also underscore the need for tailored benchmarks in distributed learning and parameter server optimizations. By aligning benchmarking efforts with evolving research needs, the field can continue to advance meaningfully and sustainably.

Energy Efficiency: As environmental and economic considerations become increasingly significant, developing energy-efficient and cost-effective solutions for distributed training has become a critical research priority. The energy demands for training large-scale models are staggering; for example, Google's Gemini Ultra incurred training costs of approximately $200M USD, while OpenAI's GPT-4 required about $80M USD [22]. Addressing these challenges effectively is essential for sustainable AI development. Recent studies reveal that a substantial portion of this energy is "lost" to unnecessary computations rather than directly contributing to training. Perseus [252] identifies these inefficiencies and proposes a system to mitigate sources of energy bloat, achieving energy consumption reductions of up to 30% without compromising training throughput or efficiency. Similarly, Argerich et al. [253] focus on measuring energy consumption during model inferencing, providing insights into energy efficiency across different deployment scenarios. Zeus [254] introduces an online exploration-exploitation approach, combined with just-in-time energy profiling, to automatically determine optimal job and GPU-level configurations for recurring deep neural network (DNN) training tasks. This adaptive scheme significantly enhances energy efficiency without degrading performance. Another innovative approach, EnvPipe [255], leverages idle time ("bubbles") created during pipeline parallelism by strategically scheduling pipeline units to align these bubbles with reduced frequency operations. This method achieves up to 25% energy savings while maintaining training efficiency. These advancements underscore the importance of optimizing both training and inference processes to minimize energy waste, demonstrating that substantial energy savings are achievable through innovative system designs and adaptive strategies.

Closing Remarks

This thesis demonstrates that marrying system-level insight with data-centric scheduling yields *both* faster *and* more reliable distributed training. By stepping beyond one-off optimisations and treating architecture, protocol, and data as a unified design space, the thesis lays a foundation for the next generation of **data-aware Parameter Servers**. It is believed that the methods and empirical evidence presented here will inform—and inspire—future work toward ever-larger, ever-more-efficient learning at scale.

Chapter 8

Publications

This chapter outlines any research work published with the participation of the author of this document. Publications are organized and presented by decreasing chronological order.

2025

- **N. Provatas**, I. Konstantinou and N. Koziris, **A Survey on Parameter Server Architecture: Approaches for Optimizing Distributed Centralized Learning** In IEEE Access, vol. 13, pp. 30993-31015, 2025
- **N. Provatas**, I. Chalas, I. Konstantinou and N. Koziris, **Strategy-Switch: From All-Reduce to Parameter Server for Faster Efficient Training**, In IEEE Access, vol. 13, pp. 9510-9523, 2025

2024

- N. Chalvantzis, A. Vontzalidis, E. Kassela, A. Spyrou, N. Nikitas, N. Provatas, I. Konstantinou, N. Koziris.**IW-NET BDA: A Big Data Infrastructure for Predictive and Geotemporal Analytics of Inland Waterways** In IEEE Access, vol. 12, pp. 52503-52523, 2024

2021

- A. Krisilias, **N. Provatas**, I. Konstantinou and N. Koziris. **A Performance Evaluation of Distributed Deep Learning Frameworks on CPU Clusters Using Image Classification Workloads.** In proceedings of the Fifth IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications (BPOD 2021 in conjuction with IEEE BigData 2021), December 2021
- **N. Provatas**, I. Konstantinou and N. Koziris. **Is Systematic Data Sharding able to Stabilize Asynchronous Parameter Server Training?** In

proceedings of the 2021 IEEE International Conference on Big Data (Big-Data 2021), December 2021

- **N. Provatas**. **Exploiting data distribution in distributed learning of deep classification models under the parameter server architecture.** Proceedings of the VLDB 2021 PhD Workshop (VLDB-PhD 2021), Copenhagen, Denmark. Vol. 2971. 2021.

2020

- **N. Provatas**, E. Kassela, N. Chalvantzis, A. Bakogiannis, I. Giannakopoulos, I. Konstantinou and N. Koziris. **SELIS BDA: Big Data Analytics for the Logistics Domain.** In proceedings of the 2020 Applications of Big Data Technology in the Transport Industry Workshop (in conjuction with IEEE BigData 2020), December 10-13 2020

2019

- E. Kassela, **N. Provatas**, I. Konstantinou, A. Floratou and N. Koziris. **General-Purpose vs. Specialized Data Analytics Systems: A Game of ML & SQL Thrones** In proceedings of the 2019 IEEE International Conference on Big Data (BigData 2019), Los Angeles, CA, USA December 9-12 2019

- E. Kassela, **N. Provatas**, A. Tsiourvas, I. Konstantinou, and N. Koziris. **BigOptiBase: Big Data Analytics for Base Station Energy Consumption Optimization** In proceedings of the 2019 IEEE International Conference on Big Data (BigData 2019), Los Angeles, CA, USA December 9-12 2019

- **N. Provatas**, I. Konstantinou and N. Koziris. **Towards Faster Distributed Deep Learning Using Data Hashing Techniques** In proceedings of the 2019 IEEE International Conference on Big Data (BigData 2019), Los Angeles, CA, USA December 9-12 2019

Appendix A

Background on Machine Learning

Deep neural networks [9] aim to approximate some function $f : \mathbb{R}^n \to \mathbb{R}$. In the context of classification problems, this function is used to classify a feature vector $\vec{x}$ to a category y, i.e. $y = f(\vec{x}; \vec{w})$, where $\vec{w}$ is used to denote the neural network parameters. Neural networks are organized as a sequence of layers, which are sequentially connected. Thus, the output of one layer becomes the input of the following one. Depending on the task, different layer types have been proposed, as convolutional layers for image classification [1].

Using neural networks for classification, requires that the neural network is trained to provide the best approximation of the aforementioned function f in respect to its parameters $\vec{w}$. The set of model parameters $\vec{w}$ that can provide such approximation is the minimizer of a *loss function*, which describes the error of a model given a set of example feature vectors $\vec{x}_1, \vec{x}_2, ..., \vec{x}_n$ and the corresponding predictions $y_1, y_2, ..., y_n$. Since the minimizer of a function is needed, finding the best network parameters is actually an optimization problem on the loss function.

Gradient Descent is considered to be among the most popular algorithm used in optimization problems [256]. However, considering both the size of deep neural networks and the vast amount of data usually available, Gradient Descent is not preferred, since each iteration will be too slow. A quicker approach is iits stochastic version, the Stochastic Gradient Descent (SGD). Therefore, this set of parameters $\vec{w}$ is located by the SGD algorithm.

Let $L : \mathbb{R}^n \to \mathbb{R}$ be the loss function used on training a model with parameters $\vec{w}$. The k-th iteration of SGD [257], based on a randomly selected training example $(\vec{x}_k, y_k)$, is mathematically formulated as follows:

$$\vec{w}_{k+1} = \vec{w}_k - \alpha \cdot \nabla_{\vec{w}_k} L(\vec{w}_k; \vec{x}_k, y_k) \tag{A.1}$$

where w_i and a stand for the model parameters in the i-th iteration of the SGD and the learning rate respectively.

SGD is more commonly used in the form of mini-batch SGD. In mini-batch SGD, a random set of n training examples $(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n)$ is used to

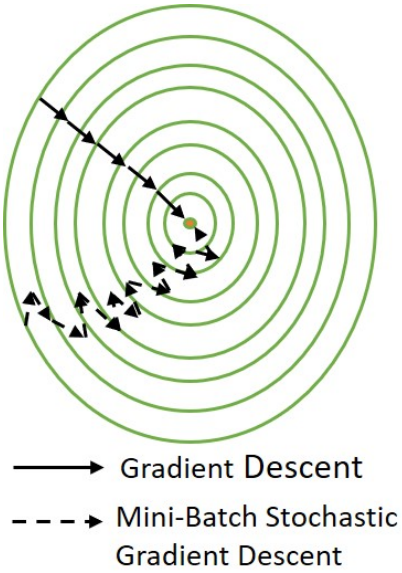


Figure A.1: Contour Plot outlining Gradient Descent and Mini-Batch Gradient Descent while converging to a local minimum point.

update the model parameters in each training iteration. The update is performed based on the average of the gradients computed by each training example and equation A.1 is rewritten as follows:

$$\vec{w}_{k+1} = \vec{w}_k - \frac{\alpha}{n} \cdot \sum_{i=1}^{n} \nabla_{\vec{w}_k} L(\vec{w}_k; \vec{x}_i, y_i) \tag{A.2}$$

The number of training examples that the mini-batch consists of is called *mini-batch size*.

While an iteration of Mini-Batch SGD is still faster than the one of GD, it is important to note that it usually needs more iterations to converge. Figure A.1 presents a contour plot, which outlines how the gradients move the weights towards the optimization point. Gradient Descent takes into account the whole data distribution in each training step and thus continuously moves towards the optimization point. However, mini-batch SGD computes the gradient of a training step with only B examples, directing the weights to various directions before converging. The aforementioned algorithms cannot guarantee convergence to the global minimization point, as optimization functions in neural network training are non-convex and they may stuck on local minima. Alternatives, as Adam [258], RMSProp [259] and others, have been proposed as less vulnerable to such phenomena [256].

When training neural networks, both SGD and mini-batch SGD can be accelerated by the momentum method [260], by integrating a velocity vector in the learning process.

Appendix B

Locality Sensitive Hashing

Considering that each data point can be presented as a d-dimensional vector in $\mathbb{R}^d$ and $\|s,t\|$ is a distance between the vectors $s,t \in \mathbb{R}^d$, the ϵ-approximate nearest neighbour search (ϵ-ANN) problem is mathematically formulated in Definition 4 [261].

Definition 4 (ϵ-approximate Nearest Neighbour Search) *Given a set $S \subset \mathbb{R}^d$, preprocess the set S to efficiently locate a point $q_0 \in S$, such that for any query point s:*

$$\|q_0, s\| \leq (1+\epsilon) \cdot \min_{t \in S} \|s,t\|$$

The ϵ-ANN problem can be generalized to the ϵ-approximate k-Nearest Neighbours (ϵ-kNN) problem. The generalized problem is formulated in Definition 5.

Definition 5 (ϵ-approximate k Nearest Neighbour Search) *Given a set $S \subset \mathbb{R}^d$ and any query point $s \in S$, preprocess the set S to efficiently provide a sequence of data points q_1, q_2, ..., $q_k \in S$, s.t. the point q_i is not further from the query point s than $1+\epsilon$ times the distance of s from its i-th nearest neighbour.*

One of the state-of-art methods to solve the aforementioned problems in sublinear time is the Locality Sensitive Hashing (LSH) [262, 263]. LSH uses a family of functions, named LSH family, to hash a set of data points to ensure that similar points will collide with greater probability than dissimilar points. The formulation of an LSH family is provided with the conditions given in Definition 6.

Definition 6 (Locality Sensitive Hashing (LSH) family) *A family $\mathcal{H} = \{h : S \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive for a similarity measure D if for any data points $q_1, q_2 \in S$, the following conditions are satisfied:*

- *if $\|q_1, q_2\| \leq r_1$, then $\mathbb{P}[h(q_1) = h(q_2)] \geq p_1$*

- *if* $\|q_1, q_2\| \geq r_2$, *then* $\mathbb{P}[h(q_1) = h(q_2)] \leq p_2$

A common problem when using the LSH families is that the probabilities p_1, p_2 used in Definition 6 might not create strict enough conditions to achieve a proper hashing. A common technique to overcome this problem is to concatenate hash functions from a given hash family, as defined in Definition 7. Thus, dissimilar points are more unlikely to be in the same bucket.

Definition 7 (AND-Concatenation of LSH Functions) *Given an* (r_1, r_2, p_1, p_2)*-sensitive LSH family* $\mathcal{H} : S \to U$ *and a positive integer* m*, a set of concatenated hash functions* $\mathcal{G} : S \to U^m$ *can be defined. Specifically, each hash function* $g \in \mathcal{G}$ *on a data point* $p \in S$ *is formulated as*

$$g(p) = (h_1(p), h_2(p), ..., h_m(p))$$

where $h_1, h_2, ..., h_m$ *are randomly chosen from the hash family* $\mathcal{H}$ *with replacement. Thus,* $\mathcal{G}$*, satisfies for any data points* $q_1, q_2 \in S$ *the properties*

- *if* $\|q_1, q_2\| \leq r_1$, *then* $\mathbb{P}[g(q_1) = g(q_2)] \geq p_1^m$
- *if* $\|q_1, q_2\| \geq r_2$, *then* $\mathbb{P}[g(q_1) = g(q_2)] \leq p_2^m$

and is an (r_1, r_2, p_1^m, p_2^m)*-sensitive LSH family.*

ετοολβοξ

Appendix C

Extended Abstract in Greek

C.1 Εισαγωγή

C.1.1 Κίνητρο

Την τελευταία δεκαετία, η βαθιά μηχανική μάθηση έχει αναδειχθεί ως ένα ιδιαίτερα δημοφιλές και επιδραστικό πεδίο, κυρίως λόγω της επιτυχίας της σε πολυάριθμες εφαρμογές μεγάλου όγκου δεδομένων. Τα νευρωνικά δίκτυα έχουν υιοθετηθεί για εργασίες όπως η ταξινόμηση εικόνων [1, 2], η αναγνώριση φωνής [3, 4], και η επεξεργασία φυσικής γλώσσας [5, 6], μεταξύ άλλων [7, 8]. Αυτή η υιοθέτηση οφείλεται σε μεγάλο βαθμό στη συνεχόμενη αύξηση του όγκου δεδομένων καθώς και στην διαθεσιμότητα πληθώρας υπολογιστικών πόρων.

Οι παραδοσιακοί αλγόριθμοι μηχανικής μάθησης συχνά φτάνουν σε κορεσμό απόδοσης όταν ο όγκος των δεδομένων εκπαίδευσης υπερβεί ένα συγκεκριμένο όριο [9], ενώ τα βαθιά νευρωνικά δίκτυα παρουσιάζουν συνεχή βελτίωση της απόδοσης με την αύξηση των δεδομένων [10, 11]. Όπως χαρακτηριστικά δήλωσε ένα από τα ηγετικά στελέχη του Google Brain Project, *«η αναλογία για τη βαθιά μηχανική μάθηση είναι ότι η μηχανή ενός πυραύλου είναι το μοντέλο βαθιάς μάθησης και τα καύσιμα είναι οι τεράστιες ποσότητες δεδομένων που μπορούμε να τροφοδοτήσουμε στους αλγορίθμους αυτούς»* [12]. Αυτός ο συνδυασμός μεγάλης κλίμακας δεδομένων και ισχυρής υπολογιστικής ισχύος έχει φέρει τη βαθιά μάθηση στην πρώτη γραμμή πολλών πεδίων.

Ένα χαρακτηριστικό παράδειγμα της ταχείας ανάπτυξης του deep learning είναι η αρχιτεκτονική ResNet της Microsoft [13], η οποία παρουσιάστηκε το 2015 και πέτυχε top-1 accuracy 78% στο σύνολο δεδομένων ImageNet [14] . Πιο πρόσφατα μοντέλα περιλαμβάνουν εκατοντάδες εκατομμύρια ή ακόμα και δισεκατομμύρια παραμέτρους και αγγίζουν ή ξεπερνούν το 90% ακρίβειας στο ImageNet [15–19].Περίληψη σημαντικών μοντέλων της τελευταίας δεκαετίας παρέχεται στον Πίνακα Γ′.1. Την ίδια χρονιά με το ResNet, η Digital Reasoning πρότεινε ένα δίκτυο με 160 δισεκατομμύρια παραμέτρους [20]. Τα μοντέλα βαθιάς μηχανικής μάθησης έχουν εξελιχθεί από αρχιτεκτονικές με δεκάδες εκατομμύρια παραμέτρους σε συστήματα αιχμής με εκατοντάδες δισε-

Πίνακας Γ′.1: Βασικά Μοντέλα Βαθιάς Μάθησης και τα Χαρακτηριστικά τους (2015–2025)

Model	**Year**	**Parameters**	**Performance**	**Proposed By**	**Ref.**
ResNet-152	2015	∼60M	Top-1: 78.57% (ImageNet)	He et al.	[13]
ResNeXt-101	2017	∼44M	Top-1: 78.8% (ImageNet)	Xie et al.	[23]
BERT-Large	2018	340M	GLUE: 80.5	Devlin et al.	[24]
GPT-2	2019	1.5B	Perplexity: 18.34 (WikiText-2)	Radford et al.	[25]
T5-11B	2019	11B	GLUE: 89.7	Raffel et al.	[26]
ViT-L/16	2020	307M	Top-1: 85.59% (ImageNet)	Dosovitskiy et al.	[27]
GPT-3	2020	175B	LAMBADA: 76.2% (few-shot)	Brown et al.	[21]
LLaMA 65B	2023	65B	Beats GPT-3 on multiple NLP benchmarks	Touvron et al.	[28]
Falcon-180B	2023	180B	Near PaLM-2 Large performance	Almazrouei et al.	[29]
Mistral 7B	2023	7B	Beats LLaMA-13B on reasoning/math/code	Jiang et al.	[30]
GPT-4	2023	∼1.8T (est.)	Bar Exam (top 10%)	OpenAI	[31]
Claude 2	2023	Not disclosed	MMLU: 78.5 (5-shot)	Anthropic	[32]
Gemini 1.5 Pro	2024	Not disclosed	1M token context window	Google DeepMind	[33]
Gemini 2.0 Flash	2025	Not disclosed	2× speed, strong multimodal	Google DeepMind	[34]

κατομμύρια ή ακόμα και τρισεκατομμύρια παραμέτρους. Το GPT-4 αποτελεί κορυφαίο παράδειγμα αυτής της τάσης [21, 22]. Τα τελευταία χρόνια, παρατηρείται επιτάχυνση στην ανάπτυξη μεγάλων γλωσσικών μοντέλων (LLMs), όπως τα Claude 2, Gemini, Falcon, LLaMA και Mistral — το καθένα με διαφορετικές στρατηγικές ως προς την κλιμάκωση, την εκπαίδευση και την προσβασιμότητα (βλ. Πίνακα Γ′.1).

Παρά τα εντυπωσιακά κέρδη απόδοσης αυτών των μεγάλων μοντέλων, η εκπαίδευσή τους απαιτεί όλο και περισσότερους υπολογιστικούς πόρους. Σύμφωνα με την έκθεση AI Index του Stanford για το 2024 [22], το Gemini Ultra της Google απαίτησε περίπου 100 δισεκατομμύρια petaFLOPS (με εκτιμώμενο κόστος $200M), ενώ το GPT-4 περίπου 10 δισεκατομμύρια petaFLOPS ($80M). Οι σύγχρονοι επιταχυντές, όπως οι GPUs, είναι κρίσιμοι για να καταστήσουν εφικτή την εκπαίδευση νευρωνικών δικτύων [35, 36]. Ωστόσο, καθώς τα δεδομένα και τα μοντέλα μεγαλώνουν, η εκπαίδευση σε ένα μόνο μηχάνημα γίνεται απαγορευτικά χρονοβόρα, οδηγώντας στην υιοθέτηση κατανεμημένων μεθόδων εκπαίδευσης νευρωνικών δικτύων [37].

Για την αντιμετώπιση αυτής της πολυπλοκότητας, η κατανεμημένη βαθιά μηχανική μάθηση ακολουθεί είτε τον *παραλληλισμό μοντέλου - model parallelism* [38,

39] είτε τον *παραλληλισμό δεδομένων - data parallelism* [40, 41]. Στον παραλληλισμό μοντέλου, τα επιπέδα ή οι παράμετροι ενός μοντέλου κατανέμονται σε πολλαπλά μηχανήματα. Αντιθέτως, στον παραλληλισμό δεδοομένων το σύνολο δεδομένων διαιρείται σε τμήματα, το καθένα ανατίθεται σε ένα μηχάνημα εργάτη, και όλοι εκπαιδεύουν το ίδιο καθολικό μοντέλο σε διακριτά υποσύνολα δεδομένων. Ο παραλληλισμός δεδομένων έχει κυριαρχήσει λόγω της ανεξαρτησίας του από την αρχιτεκτονική και της δυνατότητας να εκμεταλλευτεί μεγάλα σύνολα δεδομένων για καλύτερη απόδοση. Άλλη γνωστή στρατηγική είναι ο *παραλληλισμός διοχέτευσης- pipeline parallelism* [42–44], όπου μικρές ομάδες δεδομένων περνούν σειριακά από επιμέρους τμήματα του μοντέλου. Παράλληλα χρησιμοποιούνται και υβριδικές προσεγγίσεις [44] των παραπάνω μεθόδων.

Πέρα από τη στρατηγική παραλληλισμού, ένας ακόμη βασικός παράγοντας στις κατανεμημένες τεχνικές μάθησης είναι η επιλογή αρχιτεκτονικής επικοινωνίας. Οι εργάτες μπορούν να επικοινωνούν με *μηχανισμούς ολικής μείωσης - all-reduce*, π.χ. του δαχτυλιδιού, του δέντρου ή άλλες ιεραρχικές προσεγγίσεις. Άλλη επιλογή είναι τα αποκεντρωμένα δίκτυα, όπως αυτό των ομότιμων κόμβων. Μια πολύ διαδεδομένη προσέγγιση είναι η κεντρική αρχιτεκτονική του *εξυπηρετή παραμέτρων - parameter server*.Σε αυτό το μοντέλο, πολλοί εργάτες υπολογίζουν διανύσματα κλίσεων πάνω σε διακριτά τεμάχια δεδομένων και τα στέλνουν σε κεντρικούς εξυπηρετητές παραμέτρων, οι οποίοι διατηρούν το κεντρικό μοντέλο. Η αρχιτεκτονική αυτή είναι κατάλληλη για μεγάλα δίκτυα και σύνολα δεδομένων.

Από τις διάφορες διαθέσιμες προσεγγίσεις για κατανεμημένη βαθιά μηχανική μάθησης που παρουσιάστηκαν, η παρούσα διατριβή εστιάζει στο χαρακτηριστικό παράδειγμα του παράλληλου ως προς τα δεδομένα *εξυπηρετητή παραμέτρων* [39, 57–59]. Αξίζει να σημειωθεί ότι η αρχιτεκτονική αυτή μπορεί να υποστηρίξει και παραλληλισμό μοντέλου ή υβριδικές στρατηγικές, οι οποίες όμως δεν αποτελούν μέρος της παρούσας μελέτης. Η διατριβή ξεκινά παρουσιάζοντας την ανάγκη για εξειδικευμένες αρχιτεκτονικές όπως ο εξυπηρετητής παραμέτρων, και παρουσιάζει δύο συγκεκριμένες βελτιστοποιήσεις για αυτή την αρχιτεκτονική.

C.1.2 Κύριες Συνεισφορές

Οι κύριες συνεισφορές αυτής της διατριβής είναι οι εξής:

1. Γίνεται αξιολόγηση της απόδοσης της εξειδικευμένης αρχιτεκτονικής του εξυπηρετητή παραμέτρων σε σύγκριση με ένα γενικό σκοπού μοντέλο επεξεργασίας, όπως το MapReduce. Η απόδοση διαφέρει κατά μέσο όρο κατά 8.23× όταν κλιμακώνεται σε 140 κόμβους.

2. Παρουσιάζεται μια συστηματική βιβλιογραφική ανασκόπηση της έρευνας γύρω από την αρχιτεκτονική του εξυπηρετή παραμέτρων.

3. Εισάγεται η τεχνική Εναλλαγής Στρατηγικής (Strategy-Switch), που χρησιμοποιεί All-Reduce για να ξεκινήσει την εκπαίδευση με τον εξυπηρετήτη παραμέτρων από ένα καλύτερο σημείο ελαχίστου.

4. Συζητούνται στρατηγικές διανομής δεδομένων για τη βελτίωση της εκπαίδευσης με εξυπηρετητή παραμέτρων.

5. Μετράται η επίδραση της συστηματικής κατανομής δεδομένων έναντι της τυχαίας. Η χρήση συστηματικής κατανομής μειώνει έως και 8× την διακύμανση στις μετρικές εκπαίδευσης και 2× επικύρωσης.

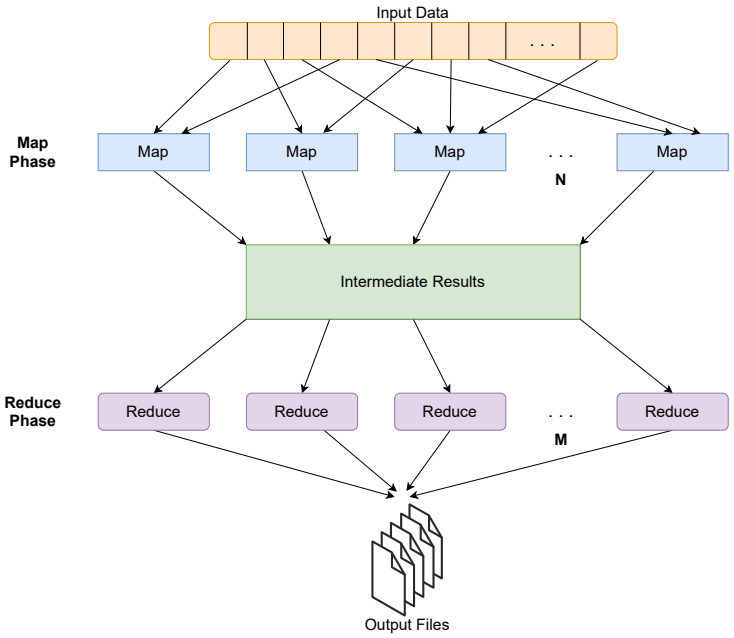
C.2 Κατανεμημένες Αρχιτεκτονικές: Από γενικού σκοπού στις εξειδικευμένες αρχιτεκτονικές μηχανικής εκμάθησης

Η συνεχής αύξηση του μεγέθους των διαθέσιμων δεδομένων οδήγησε τους ερευνητές να προτείνουν διάφορες αρχιτεκτονικές συστημάτων που θα είναι σε θέση να χειριστούν μεγάλο όγκο δεδομένων. Τέτοιες αρχιτεκτονικές μπορεί να είναι γενικού σκοπού και να είναι εφαρμόσιμες σε μια πληθώρα διαφορετικών τύπων εργασιών. Ένα χαρακτηριστικό προγραμματιστικό μοντέλου γενικού σκοπού είναι το Map-Reduce,που υιοθετείται από διάφορα συστημάτα γενικού σκοπού. Ωστόσο, ανάλογα με τον τύπο του φόρτου εργασίας, έχουν προταθεί άλλες εξειδικευμένες αρχιτεκτονικές. Στην περίπτωση των μοντέλων μάθησης με κατανεμημένο τρόπο, έχουν προταθεί πολλαπλές προσεγγίσεις, με τον διακομιστή παραμέτρων να είναι ένας από μία από τις πιο εξέχουσες. Σε αυτό το κεφάλαιο, παρουσιάζεται το Map-Reduce και πώς μπορεί να εφαρμοστεί στο πλαίσιο προβλημάτων μάθησης. Επιπλέον, παρέχεται ένα λεπτομερές υπόβαθρο σχετικά με την προσέγγιση του διακομιστή παραμέτρων.

C.2.1 Map-Reduce: Γενικά και εφαρμογές στη μηχανική μάθηση.

Το 2004, η Google [67] πρότεινε το μοντέλο προγραμματισμού Map-Reduce για εφαρμογές επεξεργασίας μεγάλων δεδομένων. Οι εφαρμογές που εκμεταλλεύονται αυτό το μοντέλο προγραμματισμού αποτελούνται από ένα σύνολο συναρτήσεων *αντιστοίχισης* - *map* και *μείωσης* - *reduce*. Η Εικόνα Γ΄.1 παρουσιάζει το προγραμματιστικό μοντέλο του Map-Reduce. Τα δεδομένα εισόδου / εξόδου συνήθως αποθηκεύονται σε κάποιο κατανεμημένο σύστημα αρχείων, όπως το Apache Hadoop [69, 70].

Το προγραμματιστικό μοντέλο του Map-Reduce συζητήθηκε σε προβλήματα μάθησης, όπου αποδεικνύεται ότι υπό τις προϋποθέσεις του στατιστικού μοντέλου ερωτημάτων (π.χ. αλγορίθμων K-Κέντρων, αλγόριθμος ανάστροφης διάδοσης στα νευρωνικών δικτύων), ένα πρόβλημα μάθησης μπορεί να γραφεί σε μία περιληπτική μορφή κατάλληλη για επεξέργασία μέσω του Map-Reduce [73, 77, 78].



Εικόνα Γ΄.1: Το προγραμματιστικό μοντέλο Map-Reduce.

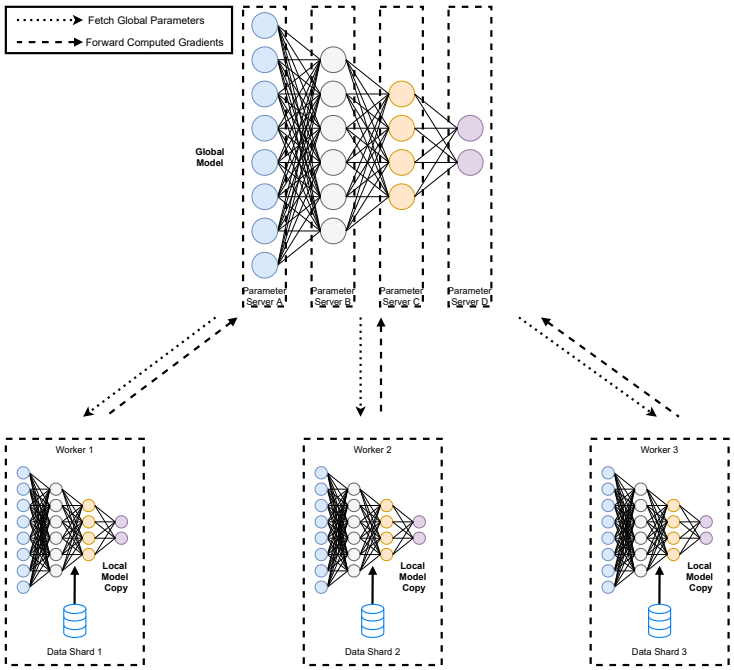
C.2.2 Κατανεμημένες αρχιτεκτονικές μάθησης

Παρά τις διάφορες προτεινόμενες προσεγγίσεις, η επαναληπτική προσέγγιση αυτών των αλγορίθμων έρχεται σε αντίθεση με τη φύση του παραδείγματος MapReduce [79]. Ως εκ τούτου, έχει προκύψει η ανάγκη για εξειδικευμένες αρχιτεκτονικές κατάλληλες για τον τομέα μηχανικής μάθησης, οι οποίες ακολουθούν δύο διαφορετικούς τύπους παραλληλισμού: τον παραλληλισμό δεδομένων και τον παραλληλισμό μοντέλων. Ο παραλληλισμός μοντέλων [38, 39] υιοθετείται στην περίπτωση εξαιρετικά μεγάλων μοντέλων που δεν μπορούν να χωρέσουν στη μνήμη ενός μεμονωμένου μηχανήματος που πρόκειται να εκπαιδευτεί. Ο παραλληλισμός δεδομένων [40] χρησιμοποιείται για τη διαχείριση του αυξανόμενου όγκου των δεδομένων, όπου τα δεδομένα χωρίζονται σε τμήματα που έχουν εκχωρηθεί σε διαφορετικές υπολογιστικές δομές μίας συστοιχίας υπολογιστών. Σήμερα, πολλαπλά κατανεμημένα συστήματα βαθιάς μάθησης υποστηρίζουν τον παραλληλισμό δεδομένων, ενώ μπορεί να εφαρμοστεί με τον ίδιο τρόπο ανεξάρτητα από το μοντέλο που χρησιμοποιείται για την εκπαίδευση [41].

Με το μοντέλο παραλληλισμού δεδομένων, η κατανεμημένη εκπαίδευση μπορεί να εκτελεστεί κάτω από δύο διαφορετικές αρχιτεκτονικές. Μια προσέγγιση βασίζεται στη χρήση τεχνικών All-Reduce [80] σε ένα peer-to-peer δίκτυο κόμβων [81–83]. Μια άλλη προσέγγιση έγκειται στη διάκριση των κόμβων μεταξύ δύο διαφορετικών ομάδων, των διακομιστών και των εργαζομένων, ακολουθώντας την αρχιτεκτονική του διακομιστή ή εξυπηρετητή παραμέτρων. Σε αυτή την διατριβή, εστιάζουμε στη μελέτη και τη βελτιστοποίηση της αρχιτεκτονικής του διακομιστή παραμέτρων, την οποία θα συζητήσουμε εκτενώς στην επόμενη ενότητα.

C.2.3 Η αρχιτεκτονική διακομιστή παραμέτρων

Ο διακομιστής παραμέτρων [39, 57–59, 84, 85] είναι μια ευρέως χρησιμοποιούμενη παράλληλη αρχιτεκτονική δεδομένων, η οποία μπορεί να ακολουθηθεί για την εκπαίδευση μοντέλων βαθιάς εκμάθησης με κατανεμημένο τρόπο.



Εικόνα Γ΄.2: Η αρχιτεκτονική διακομιστή παραμέτρων.

Η Εικόνα Γ΄.2 περιγράφει τη διαδικασία εκπαίδευσης στην αρχιτεκτονική διακομιστή παραμέτρων. Οι διακομιστές έχουν μοιραστεί ένα καθολικό μοντέλο και ο καθένας είναι υπεύθυνος για την ενημέρωση του αντίστοιχου μέρους του καθολικού μοντέλου μοντέλου. Το σετ εκπαίδευσης μοιράζεται μεταξύ των συμμετεχόντων εργαζομένων στη διαδικασία, όπου ο καθένας έχει ένα τοπικό αντίγραφο του μοντέλου για να υπολογίσει ένα σύνολο διανυσμάτων κλίσεων χρησιμοποιώντας το υποσύνολο των δεδομένων που τους έχει εκχωρηθεί. Τα διανύσματα κλίσεων ωθούνται πίσω στους διακομιστές, οι οποίοι με τον επιλεγμένο αλγόριθμο βελτιστοποίησης θα ενημερώσουν το καθολικό μοντέλο. Στην αρχή κάθε επανάληψης, οι εργαζόμενοι ζητάνε την επικαιροποιημένη έκδοση του καθολικού μοντέλου.

Η ρύθμιση των υπερπαραμέτρων εκπαίδευσης του δικτύου πραγματοποιείται αυτόματα, όσον αφορά το μέγεθος της μικρο-ομάδας που θα χρησιμοποιεί ο κάθε εργαζόμενος σε κάθε επανάληψη και το ρυθμό μάθησης [88], σύμφωνα με τις υπερπαραμέτρους εκπαίδευσης σε ένα κόμβο.

Αναφορικά με το συγχρονισμό, στη βασική εκδοχή της, η αρχιτεκτονική διακομιστή παραμέτρων μπορεί να πραγματοποιήσει την εκπαίδευση είτε με πλήρη συγχρονισμό μεταξύ των εργαζομένων σε κάθε επανάληψη είτε με πλήρη απουσία αυτού. Στην

Πίνακας Γ′.2: TensorFlow vs. Spark MLlib

	TensorFlow	Σπαρκ ΜΛλιβ
Αφηρημένο Προγραμματιστικό Μοντέλο	DAG	ΔΑΓ
Μοντέλο Εκτέλεσης	Μακράς διάρκειας εργασίες διακομιστών και εργαζόμενων	Σύντομες εργασίες αντίστοιχισης και μείωσης στο Spark
Συγχρονισμός	Σύγχρονη και ασύγχρονη εκπαίδευση	Σύγχρονη εκπαίδευση
Πρόσβαση στα Δεδομένα	Αποφυγή μεταφοράς άχρηστων δεδομένων από το δίσκο / Caching	Μεταφορά όλου του σύνολου εκπαίδευσης από το δίσκο / Caching

απουσία συγχρονισμού ενδέχεται οι εργαζόμενοι να χρησιμοποιούν διαφορετικές παραμέτρους στο ίδιο βήμα για τον υπολογισμό των διανυσμάτων κλίσεων.

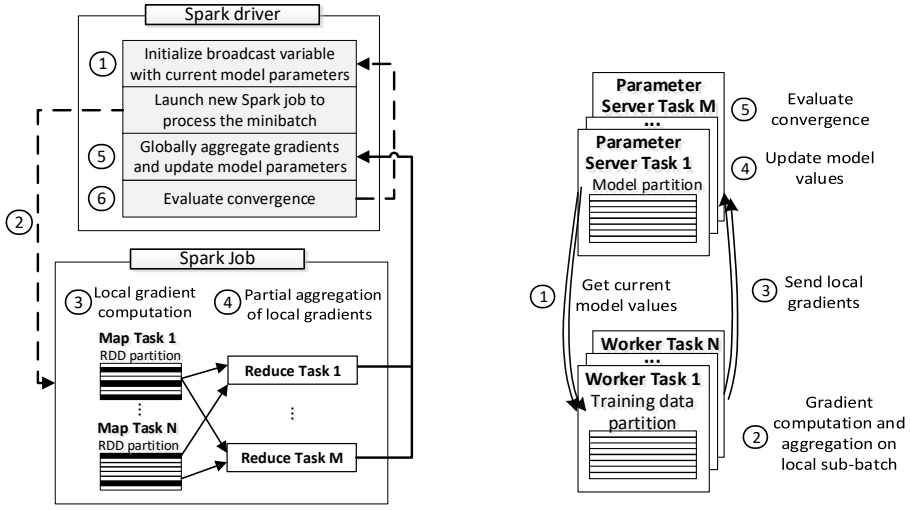
C.3 Γιατί είναι απαραίτητες οι ειδικές αρχιτεκτονικές: Μία συγκρητική αξιολόγηση του Map-Reduce με τον Διακομιστή Παραμέτρων σε μεγάλη κλίμακα

Σύμφωνα με όσα συζητήθηκαν παραπάνω, τόσο αρχιτεκτονικές γενικού όσο και εξειδικευμένου σκοπού μπορούν να χρησιμοποιηθούν για την κατανεμημένη εκπαίδευση μοντέλων. Σε αυτήν την ενότητα γίνεται μία συγκρητική αξιολόγηση του Map-Reduce με τον διακομιστή παραμέτρων. Για την σύγκριση χρησιμοποιούνται συστήματα που αξιοποιούν τις αρχιτεκτονικές: το Apache Spark για το Map-Reduce και το Google TensorFlow για τον διακομιστή παραμέτρων.

C.3.1 Αρχιτεκτονικές Συστημάτων

Ο Πίνακας Γ′.2 παρουσιάζει τις βασικές διαφορές των δύο σύστηματων [63, 264]. Τα δύο συστήματα ακολουθούν ένα αφηρημένο προγραμματιστικό μοντέλο που στηρίζεται σε κατευθυνόμενους ακυκλικούς γράφους. Ιδιαίτερο ενδιαφέρον παρουσιάζεται στα μοντέλα εκτέλεσης των δύο συστημάτων, τα οποία παρουσιάζονται αναλυτικά στην Εικόνα Γ′.3.

Στην περίπτωση του TensorFlow (Εικόνα Γ′.3β′), τα βήματα 1-3 και 4-5 εκτελούνται παράλληλα στις εργασίες του εργάτη και του διακομιστή παραμέτρων αντίστοιχα. Κάθε εργαζόμενος λαμβάνει τα πιο πρόσφατα παραμέτρους μοντέλου από τον διακομιστή παραμέτρων (Βήμα 1), εκτελεί τον υπολογισμό του τοπικού διανύσματος κλίσης (Βήμα 2) και στη συνέχεια στέλνει το σχετικό διάνυσμα στους διακομιστές παραμέτρων (Βήμα 3). Ακολούθως, οι διακομιστές παραμέτρων ενημερώνουν τις παραμέτρους του μοντέλου με τα διανύσματα κλίσεων που έλαβαν (Βήμα 4) και αξιολογούν



(α′) Μοντέλο Εκτέλεσης Spark (β′) Μοντέλο Εκτέλεσης TensorFlow

Εικόνα Γ′.3: Μοντέλα Εκτέλεσης Spark και TensorFlow

τη σύγκλιση του μοντέλου (Βήμα 5).

Στην περίπτωση του Spark (Εικόνα Γ′.3), οι αλγόριθμοι μηχανικής μάθησης, που εκτελούνται μέσω της βιβλιοθήλης MLlib αναπτύσσονται ως μια ακολουθία εργασιών Spark που αποτελούνται από στάδια *map* και *reduce*. Η κατάσταση της εκπαίδευσης διατηρείται σε όλες τις εργασίες μέσω του *Spark driver* που είναι το κύριο πρόγραμμα της εφαρμογής Spark. Ο Spark driver χρησιμοποιεί τις μεταβλητές *broadcast* [92] για να μεταφέρει την κατάσταση στις εργασίες Spark που έχουν ξεκινήσει. Τα δεδομένα εκπαίδευσης αντιπροσωπεύονται συνήθως ως ένα διαμερισμένο RDD που είναι μια συλλογή στοιχείων που μπορούν να λειτουργήσουν παράλληλα [93]. Στο πλαίσιο του αλγορίθμου SGD , εκκινείται μια ακολουθία εργασιών Spark Γ′.3, καθεμία από τις οποίες επεξεργάζεται μια μικρό-ομάδα δεδομένων, μέχρι να συγκλίνει ο αλγόριθμος. Πρώτον, ο Spark driver προετοιμάζει μια broadcast μεταβλητή που περιέχει τις τρέχουσες τιμές των παραμέτρων του μοντέλου (Βήμα 1). Στο επόμενο βήμα, ξεκινά μια εργασία Σπαρκ η οποία είναι υπεύθυνη για τη δημιουργία μιας μικρό-ομάδας, επιλέγοντας τυχαία ένα υποσύνολο παραδειγμάτων εκπαίδευσης από το κατανεμημένο RDD, και την επεξεργασία της (Βήμα 2). Οι εργασίες map υπολογίζουν τα διανύσματα κλίσεων πάνω από τα τοπικά τμήματα του RDD τους (Βήμα 3). Αφού υπολογιστούν όλες οι τοπικές κλίσεις, οι εργασίες reduce εκτελούν μια συνάθροιση των μερικών διανυσμάτων (Βήμα 4). Ο Spark driver συλλέγει την έξοδο των εργασιών reduce, εκτελεί την καθολική συνάθροιση και διατηρεί τις ενημερωμένες παραμέτρους του μοντέλου (Βήμα 5). Τέλος, ο Spark driver αξιολογεί τα κριτήρια σύγκλισης και επαναλαμβάνει τα παραπάνω βήματα εάν χρειάζεται (Βήμα 6).

Αναφορικά με την πρόσβαση στα δεδομένα, το TensorFlow αξιοποιεί το Dataset API που παράγει μικρο-ομάδες δεδομένων προς επεξεργασία, ενώ το Spark MLlib

Πίνακας Γ′.3: Χαρακτηριστικά Εικονικών Μηχανών

Χαρακτηριστικό	Τιμή
vCPUs	2
RAM	8GB
HDD	20GB
Λειτουργικό Σύστημα	Debian Jesse 8.10

Πίνακας Γ′.4: Συστήματα

	Spark	TensorFlow
Έκδοση	2.4	1.13
Κόμβος Αφέντη	1 Spark Driver	1 Διακομιστής Παραμέτρων
Ανά Κόμβο Εργάτη	1 Spark Executor	1 Κόμβο Εργάτη

Πίνακας Γ′.5: Χαρακτηριστικά Πραγματικών Δεδομένων

Σύνολο Δεδομένων	Γραμμές	Χαρακτηριστικά	Μέγεθος
Higgs	10.500.000	28	2.7GB
Year Prediction MSD	470.000	90	0.391GB

Πίνακας Γ′.6: Χαρακτηριστικά Συνθετικών Δεδομένα ανά Μοντέλο Μάθησης

Μοντέλο	Γραμμές	Χαρακτηριστικά	Μέγεθος
Λογιστική Παλινδρόμηση	980.000.000	28	252GB
Γραμμική Παλινδρόμηση	280.000.000	90	238GB
Perceptron	441.000.000	28	110GB

φορτώνει όλο το σύνολο δεδομένων με την μορφή ενός κατανεμημένου RDD.

C.3.2 Πειραματική Διάταξη

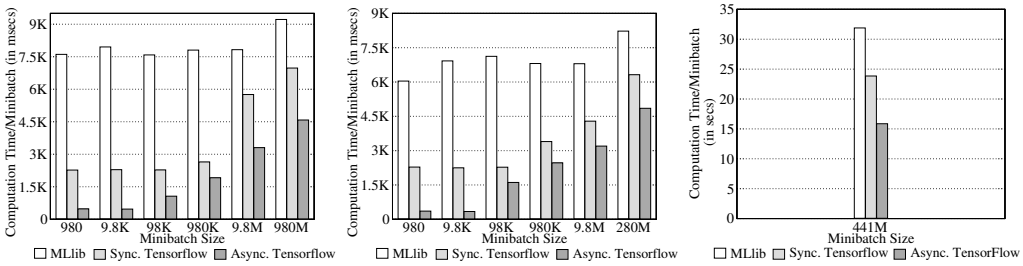
Το σύνολο των πειραμάτων εκτελούνται σε ένα σύμπλεγμα 141 εικονικών μηχανών - κόμβων στο δημόσιο σύννεφο του Okeanos [94, 95], θεωρώντας 1 ως αφέντη και 140 ως εργάτες. Τα χαρακτηριστικά των εικονικών μηχανών και των συστήματων που χρημοποιούνται δίνονται στους πίνακες Γ′.3 και Γ′.4 αντίστοιχα.

Αναφορικά με τα μοντέλα μηχανικής μάθησης χρησιμοποιούνται τρία διαφορετικά: το μοντέλο λογιστικής παλινδρόμησης, το γραμμικής παλινδρόμησης και ένα perceptron 4 επιπέδων. Εκτός του τελευταίου, που το Spark έχει διαθέσιμο μόνο τον αλγόριθμο βελτιστοποίησης καθόδου κλίσεων, στα υπόλοιπα εξετάζεται και η στοχαστική εκδοχή του με διάφορα μεγέθη μικρο-ομάδων. Η μεθοδολογία εκτέλεσης των πειραμάτων διακρίνεται σε δύο μέρη. Αρχικά, γίνεται αξιολόγηση των συστήματων σε όλη τη συστοιχία με συνθετικά δεδομένα για συγκεκριμένο αριθμό επαναλήψεων - 100 για τα μοντέλα παλινδρόμησης και 20 για το perceptron. Έπειτα, σε μία μικρή συστοιχία 5 υπολογιστών αξιολογούνται τα συστήματα αναφορικά με τη δυνατότητα σύγκλισης τους σε πραγματικά σύνολα δεδομένων. Τα δεδομένα που αξιοποιήθηκαν δίνονται στους πίνακες.

C.3.3 Πειραματική Αξιολόγηση Μεγάλης Κλίμακας σε Συνθετικά Δεδομένα

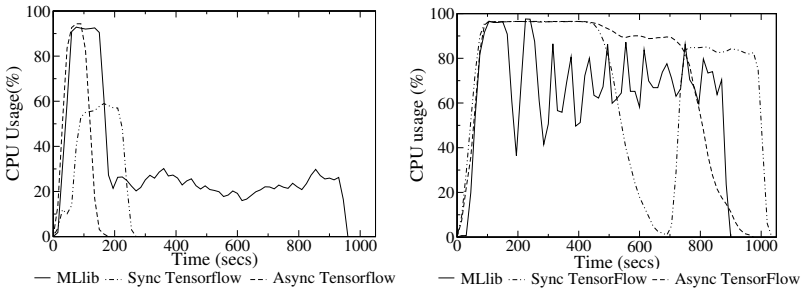
Σε αυτήν την ενότητα γίνεται παρουσιάση των ευρημάτων σχετικά με την απόδοση των συστημάτων σε μεγαλή κλίμακα, από την εκτέλεση την εκπαίδευσης των μοντέλων παλινδρόμησης και perceptron για 100 και 20 επαναλήψεις στα συνθετικά δεδομένα του πίνακα Γ′.6.

Η εικόνα Γ′.4 δείχνει τον μέσο χρόνο που αφιερώνει κάθε σύστημα για την εκτέλεση υπολογισμών διανυσμάτων κλίσεων κατά την επεξεργασία μιας μικρο-ομάδας. Παρουσιάζονται τρεις ράβδοι για κάθε μέγεθος μικρο-ομάδας, που αναφέρονται στην



(α′) Λογ. Παλινδρόμηση (β′) Γραμ. Παλινδρόμηση (γ′) Perceptron

Εικόνα Γ′.4: Χρόνος Υπολογισμού ανά Μικρό-ομάδα στην MLlib και στο TensorFlow (σύγχρονη και ασύγχρονη εκδοχή) σε μία συστοιχία 141 υπολογιστικών κόμβων.

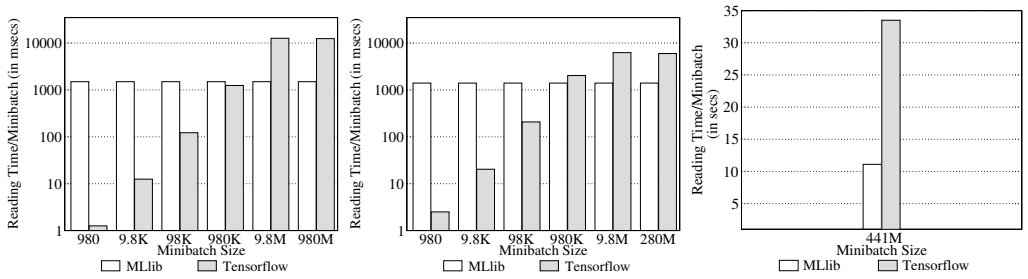


(α′) Λογ. Παλινδρόμηση (980K) (β′) Perceptron

Εικόνα Γ′.5: CPU συστοιχίας 141 κόμβων για την MLlib και το TensorFlow (σύγχρονη και ασύγχρονη εκδοχή).

Spark MLlib, στο σύγχρονο και στο ασύγχρονο Tensorflow, εκτός του perceptron που εκτελείται μόνο ο πλήρης αλγόριθμος καθόδου κλίσεων με μέγεθος μικρο-ομάδας ίσο με το μέγεθος του dataset.

Το TensorFlow ξοδεύει πάντα λιγότερο χρόνο για τον υπολογισμό κλίσεων ανεξάρτητα από τη λειτουργία εκπαίδευσης. Στα προβλήματα παλινδρόμησης (Εικόνα Γ′.4α′ και Γ′.4β′), καθώς το μέγεθος της μικρο-ομάδας μειώνεται, το TensorFlow είναι πιο αποδοτικό υπολογιστικά από το Spark MLlib.Επιπλέον, όσο μειώνεται το μέγεθος της μικρο-ομάδας, το TensorFlow επιτυγχάνει επιτάχυνση έως και $14X$, ενώ το Σπαρκ γίνεται ταχύτερο έως και $1,36X$. Σε σύγκριση με τις μακράς-διάρκειας διεργασίες του TensorFlow, το Spark λανσάρει μια νέα εργασία κάθε φορά που επεξεργάζεται μια νέα μικρο-ομάδα. Αυτό εισάγει γενικά επιπλέον κόστος προγραμματισμού εργασιών και αρχικοποίησης (ειδικά όταν το μέγεθος της μικρο-ομάδας μειώνεται και οι εργασίες χάρτη γίνονται πολύ πιο σύντομες(≈100msecs)) και οδηγεί σε χαμηλή χρήση της CPU, όπως επιβεβαιώνεται στην Εικόνα 2.8. Το ίδιο σχήμα υποδηλώνει ότι η επίδοση του σύγχρονου TensorFlow είναι μειωμένη, γεγονός που αποδίδεται κυρίως στο κόστος συγχρονισμού. Ωστόσο, εξακολουθεί να έχει καλύτερη χρήση της CPU από



(a) Λογ. Παλινδρόμηση (b) Γραμ. Παλινδρόμηση (c) Perceptron

Figure C.6: Χρόνος Ανάγνωσης ανά Μικρό-ομάδα στην MLlib και στο TensorFlow (σύγχρονη και ασύγχρονη εκδοχή) σε μία συστοιχία 141 υπολογιστικών κόμβων.

το Spark.

Η Εικόνα C.6 δείχνει τον μέσο χρόνο που αφιερώνει κάθε σύστημα στην ανάγνωση δεδομένων (απόκτηση, αποσειροποίηση και αποκωδικοποίηση ακατέργαστων βψτες) κατά την επεξεργασία μιας μικρο-ομάδας. Ο χρόνος ανάγνωσης παρουσιάζεται για τα MLlib και TensorFlow ανεξάρτητα τον τρόπο συγχρονισμού, αφού δεν εξαρτάται από αυτούς. Όσον αφορά τον μέσο χρόνο ανάγνωσης ανά μικρο-ομάδα, το Spark MLlib ξοδεύει το τον ίδιο χρόνο ανάγνωσης δεδομένων, ανεξάρτητα από το μέγεθος της μικρο-ομάδας, αφού διαβάζεται ολόκληρο το RDD σε όλες τις περιπτώσεις. Το TensorFlow από την άλλη φέρνει μόνο τα δεδομένα που την μικρο-ομάδα που απαιτείται για τον υπολογισμό του διανύσματος κλίσης. Ως αποτέλεσμα, στο TensorFlowo μέσος χρόνος ανάγνωσης μειώνεται όταν μειώνεται το μέγεθος της μικρο-ομάδας. Ωστόσο, στην περίπτωση της γραμμικής παλινδρόμησης, το Tensorflow χρειάζεται σχεδόν τον ίδιο χρόνο ανάγνωσης ανά μικρό-ομάδα τόσο για την περίπτωση του GD όσο και για αυτήν του SGD (μέγεθος μίνι παρτίδας 9,8M) όπως φαίνεται στην Εικόνα C.6b. Δεδομένου ότι κάθε αλγόριθμος παλινδρόμησης εκτελείται έως ότου το σύστημα επεξεργαστεί 100 μικρό-ομάδες, το πλήρες σύνολο δεδομένων διαβάζεται και από τους δύο αλγόριθμους.

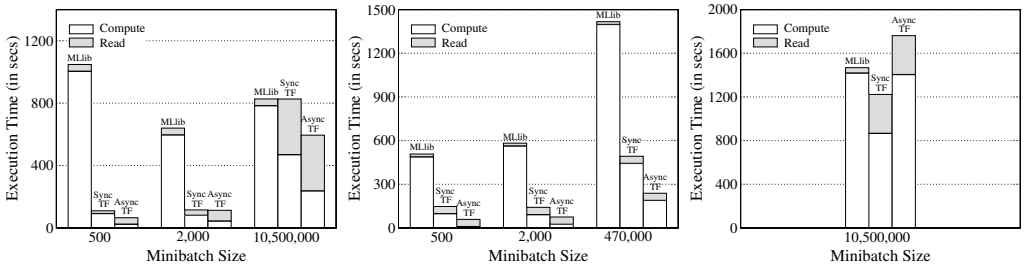
Στην περίπτωση του perceptron, δεδομένου ότι το πλήρες σύνολο δεδομένων χρησιμοποιείται ως μικρο-ομάδα (GD),το TensorFlow αφιερώνει περισσότερο χρόνο στην ανάγνωση δεδομένων από το Spark. Όπως φαίνεται στην Εικόνα Γ′.5β′, το TensorFlow, ειδικά σε σύγχρονη λειτουργία, αποκλείεται από υπολογισμούς έως ότου όλοι οι εργάτες του ολοκληρώσουν την ανάγνωση των τοπικών τους δεδομένων, (μείωση CPU μεταξύ 500 και 700 sec), γεγονός που δεν παρατηρείται στην ασύγχρονη εκδοχή.

C.3.4 Πειραματική Αξιολόγηση σε Πραγματικά Δεδομένα

Η Εικόνα Γ′.7 παρουσιάζει τον συνολικό χρόνο εκτέλεσης των TensorFlow και Spark MLlib, μοιρασμένο στον χρόνο που δαπανάται για την ανάγνωση δεδομένων

Πίνακας Γ′.7: Πλήθος μικρο-ομάδων που επεξεργάστηκε το κάθε σύστημα μέχρι τη σύγκλιση.

Μοντέλο Μηχ. Μάθησης	Μέγεθος Μικρο-ομάδας	Επεξεργασμένες Μικρο-ομάδες		
		MLlib	Σύγχρονο TF	Ασύγχρονο TF
Λογιστική Παλινδρόμηση	10,500,000	317	317	137
	2000	774	737	1006
	500	1375	1644	2449
Γραμμική Παλινδρόμηση	470,000	3104	3104	1328
	2000	2678	2773	1042
	500	2438	2540	1121
Perceptron	10,500,000	72	72	118



(α′) Λογ. Παλινδρόμηση (β′) Γραμ. Παλινδρόμηση (γ′) Perceptron

Εικόνα Γ′.7: Επίδοση της Spark MLlib και του TensorFlow (σύγχρονη και ασύγχρονη εκδοχή) για εκπαίδευση μέχρι τη σύγκλιση σε μία συστοιχία 5 υπολογιστικών κόμβων.

και την εκτέλεση υπολογισμών σε κάθε σύστημα. Τα διάφορα μεγέθη μικρο-ομάδων επιλέγονται ώστε να είναι ρεαλιστικά [102].

Αναφορικά με τα προβλήματα παλινδρόμησης, το TensorFlow είναι ταχύτερο από το Spark MLlib έως και $16X$ όταν χρησιμοποιείται ο αλγόριθμος SGD (Εικόνα Γ′.7α′, μέγεθος μικρο-ομάδας 500). Αυτή η συμπεριφορά αποδίδεται κυρίως σε αρχιτεκτονικές διαφορές των δύο συστημάτων, όπως εξηγήσαμε στην προηγούμενη ενότητα. Ανεξαρτήτως συγχρονισμού, το TensorFlow παρουσιάζει στη χειρότερη την ίδια απόδοση με το MLlib στα προβλήματα παλινδρόμησης. Επιπλέον, παρατηρείται ότι η MLlib παρουσίαζει την καλύτερη επίδοση της για μέγεθος μικρο-ομάδας 200 και όχι στην περίπτωση χρήσης του GD. Επομένως προκύπτει το συμπέρασμα ότι η καλύτερη επίδοση του Spark προκύπτει όταν τα κόστη που εισάγονται για συγχρονισμό, εκκίνηση εργασιών κλπ. δεν υπερκαλύπτουν τον χρόνο υπολογισμών.

Ο πίνακας Γ′.7 δείχνει τον αριθμό των μικρο-ομάδων που επεξεργάζεται κάθε σύστημα πριν συγκλίνει ο εκάστοτε αλγόριθμος. Ο αριθμός των μικρο-ομάδων είναι ο ίδιος μεταξύ του σύγχρονου TensorFlow και της MLlib στην περίπτωση του GD, όπως αναμένεται. Ωστόσο, παρατηρείται μια μικρή απόκλιση στην περίπτωση του SGD.

στις σύγχρονες εκτελέσεις. Αυτό οφείλεται στο γεγονός ότι κάθε σύστημα εκτελεί την επιλογή μικρο-ομάδας διαφορετικά, με αποτέλεσμα ανόμοια δεδομένα ανά μικρό-ομάδα. Επίσης, αξίζει να αναφερθεί ότι ο GD με τον SGD επεξεργάζονται διαφορετικό πλήθος μικρό-ομάδων μέχρι να συγκλίνουν.

Στην περίπτωση του perceptron η MLlib είναι πιο γρήγορη αναλογικά με το ασύγχρονο TensorFlow. Ωστόσο, αυτό οφείλεται στο ότι τα διανύσματα κλίσης ενός εργάτη μπορεί να επικαλύψουν αυτά κάποιου άλλου με αποτέλεσμα να απαιτούνται παραπάνω επαναλήψεις για σύγκλιση. Στην σύγχρονη εκδοχή του όμως το TensorFlow είναι ταχύτερο από την MLlib επιβεβαιώνοντας την υπολογιστική του υπεροχή, ειδικά σε περιπτώσεις νευρωνικών δικτύων.

C.4 Εμβαθύνοντας στην αρχιτεκτονική διακομιστή παραμέτρων: Μία έρευνα γύρω από πρόσφατες βελτιστοποιήσεις.

Η αρχιτεκτονική διακομιστή παραμέτρων αποτελεί σήμερα το κυρίαρχο πρότυπο για κλιμακούμενη εκπαίδευση βαθιών μοντέλων και προέκυψε από δύο προάγγελους: την παράλληλη αρχιτεκτονική κλειδιού-τιμής του Smola (2010) και το ασύγχρονο Hogwild! (2011). Το 2012 η Google παρουσίασε το DistBelief, καθιερώνοντας την ιδέα των αποκεντρωμένων τμημάτων παραμέτρων που ενημερώνονται από πλήθος εργατών μέσω του Downpour SGD (καταρακτώδης στοχαστικός αλγόριθμος καθόδου κλίσεων)· αυτό έθεσε τις βάσεις για βιομηχανικές πλατφόρμες όπως TensorFlow, PyTorch, MXNet, Petuum, DeepSpeed, Ray και Horovod, οι οποίες υιοθετούν τον διακομιστή παραμέτρων είτε αυτούσιο είτε σε υβριδική μορφή με χρήση δαχτυλιδιών ολικής μείωσης.

Παρά τις πρόοδους, οι υπάρχουσες βιβλιογραφικές ανασκοπήσεις καλύπτουν τμηματικά το θέμα και για αυτό στην παρούσα διατριβή έχει διενεργηθεί συστηματική βιβλιογραφική ανασκόπηση σε άρθρα που έχουν δημοσιευτεί στο χρονικό διάστημα 2010-2024, απομονώνοντας εργασίες που κατηγοριοποιούνται σε πέντε άξονες της αρχιτεκτονικής του εξυπηρετή παραμέτρων.

- έλεγχος συνέπειας
- βελτιστοποίηση δικτύου
- διαχείριση παραμέτρων
- αντιμετώπιση κόμβων με καθυστέρηση
- ανοχή σε σφάλματα

Για να διασφαλιστεί μια ολοκληρωμένη και μεθοδική επισκόπηση του πεδίου, διενεργείται μια συστηματική βιβλιογραφική ανασκόπηση (SLR) [108], η οποία περιλαμβάνει τα ακόλουθα βήματα:

1. Ορισμός ερευνητικών ερωτημάτων (RQs): Η SLR έχει σχεδιαστεί ώστε να απαντήσει σε βασικά ερευνητικά ερωτήματα που συνάδουν με τους κύριους άξονες εξέλιξης της αρχιτεκτονικής του εξυπηρετητή παραμέτρων.

- **RQ1:** Ποιες είναι οι κύριες *στρατηγικές συγχρονισμού* που έχουν προταθεί για τον *έλεγχο συνέπειας* στους εξυπηρετητές παραμέτρων·
- **RQ2:** Πώς έχουν εξελιχθεί οι *βελτιστοποιήσεις στην επικοινωνία* ώστε να βελτιώσουν την *κλιμακωσιμότητα* και την *αποτελεσματικοτητα*·
- **RQ3:** Ποιες είναι οι κύριες προσεγγίσεις για την αντιμετώπιση του *προβλήματος αργών μηχανημάτων - stragglers* και τη βελτίωση της *της ανοχής στα σφάλματα*·

2. Στρατηγική αναζήτησης βιβλιογραφίας: Οι ερευνητικές βάσεις δεδομένων (όπως π.χ.IEEE Xplore) ερωτώνται συστηματικά για την αναζήτηση ερευνητικών δημοσιεύσεων με τους ακόλουθους όρους αναζήτησης: "parameter server", "distributed machine learning", "synchronization protocols", "gradient compression" και "fault tolerance". Η αναζήτηση καλύπτει άρθρα που έχουν αξιολογηθεί από κριτές και δημοσιεύτηκαν μεταξύ 2010 και 2024, καθώς η περίοδος αυτή σηματοδοτεί την εμφάνιση και ωρίμανση της αρχιτεκτονικής του εξυπηρετητή παραμέτρων.
3. Κριτήρια ένταξης και αποκλεισμού: Καθορίστηκαν σαφή κριτήρια για την επιλογή σχετικών μελετών:

- **Κριτήρια ένταξης:** άρθρα που έχουν αξιολογηθεί από κριτές, μελέτες που εστιάζουν στην αρχιτεκτονική εξυπηρετητή παραμέτρων ή εφαρμογές αυτής με συνεισφορές στους τομείς που έχουν αναφερθεί παραπάνω.
- **Κριτήρια αποκλεισμού:** άρθρα που επικεντρώνονται αποκλειστικά στο *federated learning,* μη αγγλόφωνες δημοσιεύσεις ή άρθρα χωρίς διαθέσιμο το πλήρες κείμενο.

4. Εξαγωγή και κατηγοριοποίηση δεδομένων: Από κάθε επιλεγμένη μελέτη εξάγονται ο τίτλος, το έτος και οι κύριες συνεισφορές στο πεδίο του εξυπηρετητή παραμέτρων.

Ελέγχος Συνέπειας

Ο έλεγχος συνέπειας μοντέλου στην αρχιτεκτονική του εξυπηρετητή παραμέτρων είναι κρίσιμος για την ισορροπία μεταξύ συγχρονισμού και απόδοσης. Οι βασικές στρατηγικές βρίσκονται στα δύο άκρα:

- **BSP (Bulk Synchronous Parallel)** [90]: Απόλυτος συγχρονισμός στο τέλος κάθε επανάληψης - Η μέθοδος συνεπάγεται πλήρης συνέπεια αλλά είναι ευάλωτη σε καθυστερήσεις (στραγγλερς) [114–121].

- **ASP (Asynchronous Parallel)** [109]: Ασύγχρονες ενημερώσεις χωρίς αναμονή - Η μέθοδος αυξάνει την ταχύτητα με κόστος πιθανής ασυνέπειας (απαρχαιωμένα διανύσματα κλίσης) [58, 59, 84].

Για ενδιάμεσες λύσεις, προτάθηκαν παραλλαγές k-συγχρονισμού:

- k**-BSP** [89]: Ο εξυπηρετητής περιμένει k διαφορετικούς εργάτες - αγνοεί διανύσματα κλίσης που θα καθυστερήσουν.

- k**-Batch-BSP** [89]: Περιμένει k διανύσματα κλίσης, ανεξαρτήτως από ποιον εργάτη.

- k**-ASP** [89]: Ασύγχρονο, αλλά αξιοποιεί καθυστερημένα διανύσματα κλίσης.

- k**-Batch-ASP** [113]: Όπως παραπάνω αλλά με ευελιξία σε ομάδες.

Μία διαδεδομένη εναλλακτική είναι το **SSP (Stale Synchronous Parallel)** [45], που επιτρέπει *οριοθετημένη παλαιότητα* μεταξύ των ρολογιών των εργατών με κάποιο κατώφλι. Έχει υιοθετηθεί ευρέως [85, 128–132] και σε συνδυασμό με τις τεχνικές BSP/ASP [114, 123].

Σύμφωνα με την βιβλιογραφική ανασκόπηση άλλες σημαντικές προτάσεις σχετικά με τον έλεγχο συνέπειας αποτελούν:

- Το n**-softsync** [133]: με l εργάτες ο εξυπηρετητής περιμένει τα πρώτα l/n διανύσματα κλίσης· κάθε διάνυσμα κλίσης σταθμίζεται με ρυθμό μάθησης που σχετίζεται με το πόσο παλιό είναι το διάνυσμα. Πιο πρόσφατα διανύσματα κλίσης θα δίνουν μεγαλύτερο βάρος.

- Ο **Dutta** [134] χρησιμοποιεί ρυθμό μάθησης εξαρτώμενο από την παλαιότητα με ελάχιστη τιμή ώστε να αποτρέπεται εξαιρετικά μικρό βήμα όταν η παλαιότητα είναι μεγάλη.

- Το σύστημα **Litz** [135] μοντελοποιεί την διαδικασία εκπαίδευσης ως γράφο εργασιών με *αιτιακή συνέπεια*.

- Στο **Overlap Synchronous Parallel** (OSP) [136] κάθε εργάτης έχει νήμα επικοινωνίας και υπολογισμού· τοπικοί υπολογισμοί συσσωρεύονται μέχρι να ικανοποιηθεί συγκεκριμένο κριτήριο και κατόπιν συγχρονίζονται.

- Το **Dynamic SSP** (DSSP) [137], όπου ποικίλει δυναμικά το κατώφλι του ΣΣΠ βάσει ονλινε στατιστικών.

- Το **MLFabric** [138], το οποίο επιταχύνει το BSP προτεραιοποιώντας την αποστολή μικρών ενημερώσεων με χρήση αλγορίθμων εύρεσης της συντομότερης εργασίας, εκτός εάν μια ενημέρωση πλησιάζει το όριο παλαιότητας των τιμών.

Συγκριτική αποτελεσματικότητα: Το BSPενδείκνυται για εφαρμογές που απαιτούν μέγιστη ακρίβεια και συνέπεια, αλλά υπόκειται σε καθυστερήσεις από μηχανήματα που εισάγουν καθυστέρηση. Το ASP υπερέχει σε σενάρια όπου η ταχύτητα είναι πρωτεύουσα και η εφαρμογή αντέχει πιθανή ασυνέπεια. ToSSP προσφέρει έναν συμβιβασμό, συνδυάζοντας αυξημένη ταχύτητα με ελεγχόμενο βαθμό παλαιότητας στα διανύσματα κλίσης, απαιτώντας όμως προσεκτικό ρυθμισμό και συνεχή παρακολούθηση. Η επιλογή μηχανισμού ελέγχου συνέπειας πρέπει συνεπώς να προσαρμόζεται στις ιδιαίτερες απαιτήσεις κάθε συστήματος και εφαρμογής.

Βελτιστοποιήσεις σχετικές με το δίκτυο

Οι βελτιστοποιήσεις σχετικές με το δίκτυο στους εξυπηρετητές παραμέτρων είναι απαραίτητες για την αντιμετώπιση της συμφόρησης και την μείωση του κόστους επικοινωνίας. Οι βασικές κατηγορίες είναι:

- **Συμπίεση Βαθμίδων:** Περιλαμβάνει τεχνικές αραίωσης και κβαντοποίησης. Τεχνικές όπως Top-k αραίωση [119, 132, 139] και αποζημίωση σφάλματος [139] μειώνουν τον όγκο επικοινωνίας χωρίς απώλεια ακρίβειας. Αντίστοιχα, κβαντοποίηση χρησιμοποιείται σε δημοσιευμένες εργασίες όπως οι TernGRAD [140], QSGD [141], ATOMO [142], MQGrad [143]. Επιπλέον, έχουν προταθεί εναλλακτικές όπως η σχεδίαση - sketching [145] ή αυτοκωδικοποιητές - autoencoders [146]. Ανασκόπηση σχετικών μεθόδων υπάρχει στο [147].

- **Άθροιση διανυσμάτων κλίσης σε τοπικό επίπεδο:** Προτάθηκε στο Project Adam [84] και επεκτάθηκε στο MLFabric [138], επιτρέποντας τοπικές συσσωρεύσεις των διανυσμάτων κλίσης πριν την αποστολή τους με τον εξυπηρετητή.

- **Ενημερώσεις με προτεραίοτητα:** Ενημερώσεις και παραμέτροι προωθούνται ανάλογα με τη σημασία τους, μειώνοντας τη συμφόρηση, όπως γίνεται στο Bosen [129].

- **Τοπικά KV-stores και Caching:** Το FlexRR [114] και το FlexPS [123] ενσωματώνουν τοπική προσωρινή αποθήκευση παραμέτρων ή KV stores στους εργάτες, περιορίζοντας την επικοινωνία με τον διακομιστή.

- **Διοχέτευση και Επικάλυψη:** Το Poseidon [115],το OSP [136],και το έργο του Wang [149] επικαλύπτουν υπολογισμό και επικοινωνία για μείωση της απόκρισης, ενώ ακολουθούν πρακτικές άθροισης και παράλληλης επικοινωνίας όπως και το Project Adam.

Συγκριτική Αποτελεσματικότητα: Η συμπίεση μειώνει την επικοινωνία αλλά μπορεί να επηρεάσει τη σύγκλιση. Η άθροιση διανυσμάτων κλίσης σε τοπικό επίπεδο και η χρήση κρυφής μνήμης βελτιώνουν την επαναχρησιμοποίηση αλλά έχουν απαιτήσεις μνήμης. Η προτεραιοποίηση και η επικάλυψη ενισχύουν την αποδοτικότητα αλλά προσθέτουν πολυπλοκότητα στην υλοποίηση. Η στρατηγική επιλογής εξαρτάται από την ισορροπία μεταξύ επικοινωνιακού κόστους, υπολογιστικών πόρων και ακρίβειας.

Διαχείριση παραμέτρων

Η επιλογή της τεχνικής διαχείρισης παραμέτρων εξαρτάται από το φορτίο εργασίας και τη διαμόρφωση του συστήματος. Οι στατικές προσεγγίσεις εξισορρόπησης φόρτου, όπως η κυκλική ανάθεση (round-robin) ή οι ομοιόμορφες κατανομές, είναι απλές και αποτελεσματικές σε ομοιογενή περιβάλλοντα, αλλά δυσκολεύονται σε δυναμικά φορτία ή όταν υπάρχουν μεροληπτικά μοτίβα σε συγκεκριμένες κατηγορίες δεδομένων. Οι δυναμικές μέθοδοι, όπως οι Lapse και NuPS, διαπρέπουν σε ετερογενή περιβάλλοντα και σενάρια μεροληψίας σε συγκεκριμένη κατηγορία δεδομένων, απαιτούν όμως επιπλέον υπολογιστική ισχύ και παρακολούθηση.

Οι στρατηγικές διαχείρισης παραμέτρων κατηγοριοποιούνται σε δύο βασικούς άξονες: τεχνικές αποθήκευσης και προσεγγίσεις εξισορρόπησης των παραμέτρων. Καινοτομίες όπως η κατανομή με επίγνωση της μεροληψίας (skewness-aware partitioning) και η δυναμική επαναχώριση συμπληρώνουν τις παραδοσιακές μεθόδους, προσφέροντας μια ολοκληρωμένη εικόνα των διαθέσιμων λύσεων για τη βελτιστοποίηση ενός εξυπηρετητή παραμέτρων.

Αντιμετώπιση κόμβων με καθυστέρηση

Η αποτελεσματικότητα αυτών των προσεγγίσεων εξαρτάται σε μεγάλο βαθμό από τη διαμόρφωση του συστήματος και το ίδιο το φορτίο εργασίας. Η εκ νέου ανάθεση εργασιών, η ελαστική παραλληλία και οι προσαρμογές του μεγέθους μικρο-ομάδας που χρησιμοποιείται στην εκπαίδευση είναι ιδιαίτερα αποτελεσματικές τεχνικές σε ετερογενή περιβάλλοντα, καθώς εξισορροπούν το φορτίο δυναμικά. Η απόρριψη «καθυστερημένων» διεργασιών (straggler dropping), αν και απλή και αποδοτική, ταιριάζει καλύτερα σε ομοιογενή συστήματα με κόμβους υψηλής απόδοσης. Μέθοδοι όπως η δυναμική ρύθμιση του ρυθμού εκμάθησης προσφέρουν έναν ενδιάμεσο συμβιβασμό, αλλά ενδέχεται να επιβραδύνουν τη σύγκλιση σε ακραία σενάρια με stragglers.

Ανοχή σε σφάλματα

Τα αντίγραφα παραμέτρων (parameter replication) προσφέρουν υψηλή αξιοπιστία, συνοδεύονται όμως από μεγάλες απαιτήσεις πόρων, γεγονός που τα καθιστά κατάλληλα για κρίσιμα συστήματα. Το checkpointing επιτυγχάνει μια ισορροπία μεταξύ αξιοπιστίας και απόδοσης, αλλά μπορεί να είναι λιγότερο αποτελεσματικό σε συστήματα με συχνές αστοχίες. Προχωρημένες τεχνικές, όπως αυτή του προσεγγιστικού στιγμιοτύπου - approximate snapshotting και αυτή της φραγμένης συνέπειας - bounded consistency, είναι αποδοτικές σε πόρους, απαιτούν όμως προσεκτική υλοποίηση ώστε να μη θυσιαστεί η ακρίβεια.

C.4.1 Συνδυάζοντας τεχνικές ολικής μείωσης και ασύγχρονη εκπαίδευση στον εξυπηρετητή παραμέτρων μέσω εναλλαγής στρατηγικής εκπαίδευσης για αποδοτικότερη εκπαίδευση

Σύμφωνα με την συστηματική βιβλιογραφική ανασκόπηση που πραγματοποιήθηκε έγινε φανερό ότι η ασύγχρονη εκπαίδευση στην αρχιτεκτονική του εξυπηρετητή παραμέτρων ενδέχεται να παρουσιάσει μειωμένη ακρίβεια σε σχέση με την σύγχρονη εκδοχή της λόγω των πεπαλαιωμένων διανυσμάτων κλίσης που εμφανίζονται. Για να μειωθεί το αντίκτυπο των πεπαλαιωμένων διανυσμάτων κλίσης, στην βιβλιογραφία έχει προταθεί η ιδέα της Εναλλαγής-Συγχρονισμού (Sync-Switch), η οποία ξεκινάει την κατανεμημένη εκπαίδευση με σύγχρονο τρόπο στην αρχιτεκτονική του εξυπηρετητή παραμέτρων ώστε να οι παράμετροι του μοντέλου να προσεγγίσουν πιο κοντά στο βέλτιστο σημείο. Στο σημείο αυτό, χρησιμοποιώντας ασύγχρονη εκπαίδευση, τα διανύσματα κλίσης είναι πιο μικρά σε μέτρο με αποτέλεσμα να είναι μικρότερη η επίδραση των πεπαλαιωμένων διανυσμάτων κλίσης. Ο εντοπισμός του σημείου εναλλαγής γίνεται με συλλογή στατιστικών από διάφορες προγενέστερες εκπαίδευσης του μοντέλου. Ωστόσο, η κεντρική αρχιτεκτονική του εξυπηρετητή παραμέτρων, στη σύγχρονη εκδοχή της, εκτός του κόστους συγχρονισμού, έχει να διαχειριστεί και την αυξημένη ροή φορτίου στα πλαίσια των κεντρικών μηχανημάτων που λειτουργούν ως εξυπηρετητές. Για το σκόπο αυτό, προτείνεται ο αλγόριθμος της Εναλλαγής Στρατηγικής (Strategy-Switch), ο οποίος αντί για τη σύγχρονη εκδοχή του εξυπηρετητή παραμέτρων, χρησιμοποιεί αρχικά αποκεντροποίημενο δίκτυο, όπου οι κόμβοι επικοινωνούν μέσω τεχνικών ολικής μείωσης (All-Reduce).Κάνοντας αυτήν την επιλογή μπορούμε να μειώσουμε το κόστος συγχρονισμού. Παράλληλα, προτείνεται και ένας εμπειρικός κανόνας για την αυτόματη εναλλαγή σε ασύγχρονη εκπαίδευση, χώρις να είναι απαραίτητη η συλλογή στατιστικών εκ των προτέρων.

Συγκρητική αξιολόγηση εκπαίδευσης ολικής μείωσης και ασύγχρονου εξυπηρετητή παραμέτρων

Για την αξιολόγηση των δύο τεχνικών κατανεμημένης εκπαίδευσης, πραγματοποιήθηκε συγκριτική αξιολόγηση χρησιμοποιώντας ως πρότυπα αξιολόγησης τα εξής:

- Εκπαίδευση του δικτύου ResNet-20 στο σύνολο δεδομένων CIFAR-10 (#B1)
- Εκπαίδευση του δικτύου ResNet-32 στο σύνολο δεδομένων CIFAR-100 (#B2)

Οι μετρήσεις έδειξαν ότι το All-Reduce επιτυγχάνει τη μεγαλύτερη ακρίβεια αλλά είναι το πιο αργό, ενώ ο ασύγχρονος Parameter Server (PS) είναι ταχύτερος αλλά υποφέρει από τα πεπαλαιωμένα διανύσματα κλίσης, μειώνοντας την ακρίβεια. Επιβεβαιώνεται έτσι η αρχική θεώρηση. Σε ετερογενείς συστάδες το All-Reduce αξιοποιεί περίπου 50 % της CPU, ενώ στην ππερίπτωση του εξυπηρετητή παραμέτρων η τελευταία φτάνει περίπου στο 80%. Ωστόσο, στον εξυπηρετητή παραμέτρων διπλασιάζεται η κίνηση στο δίκτυο σε σχέση με το All-Reduce.

Μεθοδολογία

Επιβεβαιώνοντας τους αρχικούς ισχυρισμούς μέσω της πειραματικής αξιολόγησης, προτείνεται το Strategy-Switch. Για ποσοστό $\alpha\%$ των συνολικών εποχών η εκπαίδευση πραγματοποιείται μέσω All-Reduce και· κατόπιν το μοντέλο αποθηκεύεται σε κατανεμημένο σύστημα αρχείων, ώστε να συνεχίσει η εκπαίδευσης μέσω ασύχρονης προσέγγισης στον εξυπηρετητή παραμέτρων. Για να αποφευχθεί η χειροκίνητη ρύθμιση του ποσοστού $\alpha\%$, εισάγεται ένας εμπειρικός κανόνας ως προς την σφάλμα επικύρωσης: Παρακολουθείται ένα κυλιόμενο παράθυρο 5 εποχών της ποσοστιαίας μεταβολής του σφάλματος επικύρωσης (s). Όταν η τιμή του s πέφτει κάτω από το φράγμα του 1%, ενεργοποιείται η εναλλαγή — σημείο όπου επιπλέον βήματα All-Reduce θα έχουν μειωμένη απόδοση χωρίς να προσδίδουν επιπλέον στην ακρίβεια. Με χρήση του εμπειρικού κανόνα, παρέχεται μία τροποποίημενη εκδοχή του αλγορίθμου Strategy-Switch, ο αλγόριθμος Εναλλαγής Στρατηγικής Εμπειρικού Κανόνα (Empirical Rule Strategy-Switch / ER-SS), που πραγματοποιεί αυτόματα την αλλαγή από Αλλ-Ρεδυσε σε ασύγχρονη εκπαίδευση.

Πειραματική Αξιολόγηση

Τα πειράματα καλύπτουν ομογενείς (#C1)και ετερογενείς (#C2)συστάδες. Τα κύρια ευρήματα είναι τα παρακάτω:

1. **Ισορροπία μεταξύ ταχύτητας/ακρίβειας.** Στη συστάδα #C1 το Strategy-Switch συγκλίνει 1,14 φορές ταχύτερα από το All-Reduce στην εκπαίδευση με CIFAR-10 (#B1) με απώλεια ακρίβειας μόλις 0,10%. Στην περίπτωση εκπαίδευσης με CIFAR-100 (#B2) το Strategy-Switch είναι 1,10 φορές ταχύτερο με απώλεια ακρίβειας της τάξης του 0,06%.

2. **Ανθεκτικότητα ως προς α.** Με τιμές του ποσοστού α από 10% έως 90% διατηρείται ακρίβεια ανώτερη του ασύγχρονου εξυπηρετητή παραμέτρων και χρόνος εκπαίδευσης σαφώς μικρότερος του All-Reduce. Ψψηλότερες τιμές του $\alpha\%$ πλησιάζουν την ακρίβεια του All-Reduce εις βάρος του χρόνου.

3. **Δυναμική εναλλαγή έναντι σταθερού** $\alpha\%$**.** Ο εμπειρικός κανόνας (ER-SS) ισοφαρίζει ή υπερβαίνει το καλύτερο σταθερό α χωρίς πρότερη γνώση του βέλτιστου σημείου αλλαγής, επιβεβαιώνοντας την πρακτική χρησιμότητά του. Στην ετερογενή συστάδα (#C2), και στα δύο πρότυπα αξιολόγησης ο ER-SSπετύχαινει την ακρίβεια του All-Reduce,ωστόσο ολοκληρώνει την εκπαίδευση κατά 52% και 28% γρηγορότερα στα πρότυπα αξιολόγησης #B1 και #B2 αντίστοιχα.

C.5 Αξιοποιώντας την κατανομή των δεδομένων για την αρχιτεκτονική διακομιστή παραμέτρων: Ιδέες και Έννοιες

Όπως έχει αναφερθεί, η ποιότητα των μοντέλων που προκύπτουν από την ασύγχρονη εκπαίδευση επηρεάζεται από διανύσματα κλίσης που έχουν υπολογιστεί στους εργάτες από ένα παρωχημένο σύνολο παραμέτρων. Μέχρι στιγμής έχουν προταθεί λύσεις σε αυτό το φαινόμενο είτε σε αλγοριθμικό επίπεδο είτε με κατάλληλη ρύθμιση του ρυθμού μάθησης. Στην ενότητα αυτή προτείνεται η αξιοποίηση της κατανομής των δεδομένων ώστε να μπορεί να εξομαλυνθεί η επίδραση των ενημερωτικών διανυσμάτων κλίσης από παλιότερες παραμέτρους.

Ένα παράδειγμα για να εξηγηθεί καλύτερα η παραπάνω ιδέα αποτελούν σύνολα δεδομένων που έχουν ανόμοιο πλήθος δεδομένων ανά κατηγορία, όπως το σύνολο εικόνων Imagenet [14, 213]. Μέχρι τώρα τα δεδομένα ανατίθενται στους εργαζόμενους κυρίως με τυχαίο τρόπο. Η ανομοιομορφία του πλήθους δεδομένων ανά κατηγορία σε συνδυασμό με την τυχαία ανάθεση δεδομένων, μπορεί να οδηγήσει σε εργαζόμενους που να είναι επηρεασμένοι περισσότερο ή λιγότερο προς κάποια μοτίβα δεδομένων. Στην ασύγχρονη εκπαίδευση, όταν προκύψει ένα παρωχημένο σύνολο παραμέτρων, είναι πιθανό ένας εργαζόμενος να μεταφέρει τις παραμέτρους σε λάθος κατεύθυνση δεδομένου ότι μπορεί να έχει μη αντιπροσωπευτική εικόνα του σύνολου δεδομένων, λόγω του υποσυνόλου που του ανατέθηκε τυχαία.

Για την εξομάλυνση αυτού του φαινομένου, είναι πιθανό να μπορεί να αξιοποιηθεί συστηματική διαχείριση των δεδομένων στην εκπαίδευση, σύμφωνα με την κατανομή τους. Αν θεωρηθεί ως γειτονιά ένα σύνολο σημείων που περικλείονται εντός μίας σφαίρας που ορίζεται μέσω ενός κεντρικού σημείου και μίας συγκεκριμένης ακτίνας, τότε κάθε εργαζόμενος πρέπει να λαμβάνει υπόψιν του εξίσου κάθε γειτονιά. Τέτοιου τύπου γειτονιές μπορούν να προκύψουν τόσο από την αρχική κατηγοριοποίηση (διαστρωμάτωση) των δεδομένων όσο και με αλγορίθμους ή συναρτήσεις ομάδοποίησης που ανακαλύπτουν τέτοια μοτίβα (κρυφή διαστρωμάτωση). Οι παραπάνω τεχνικές έχουν χρησιμοποιηθεί σε πολλές περιπτώσης που τα δεδομένα δεν είναι καθολικά προσβάσιμα [198, 200, 217–220] σε όλους τους συμμετέχοντες εργαζόμενους.

Το παραπάνω έχει σημασία σίγουρα όσον αφορά τα δεδομένα που έχει πρόσβαση ένας εργαζόμενος, όσο και πιθανώς κατά την επιλογή μίας συγκεκριμένης μικρο-ομάδας που θα χρησιμοποιηθεί στην κάθε επανάληψη της εκπαίδευσης ενός μοντέλου. Η επιλογή δεδομένων για την μικρό-ομάδα μίας επανάληψης είναι μία διαδικάσια που έχει δοκιμαστεί αρκετά και στο παρελθόν [224, 225, 227, 228].

```
1: procedure ΔΙΑΜΟΙΡΑΣΜΟΣ ΔΙΑΣΤΡΩΜΑΤΩΣΗΣ(x, y, n)
2:                          ▷ x είναι πίνακας με πολυδιαστάτους τανυστές, κάθε ένας
                              από τους οποίους παριστάνει ένα δείγμα εκπαίδευση
3:                          ▷ y είναι ο αντίστοιχος πίνακας με τις κλάσεις των δειγ-
                              μάτων εκπαίδευσης
4:                          ▷
                              οι εργαζόμενοι απαριθμούνται με 0, 1, ..., n-1
5:     classes = unique(y)                          ▷ Εντοπισμός διαθέσιμων κλάσεων
6:     for each class in classes do
7:         x_class, y_class = in_class(x, y, class)
8:         class_size = length(y_class)
9:         for i = 0 to class_size − 1 do
10:                                          ▷ Κυκλικός διαμοιρασμός κάθε κλάσης
11:            worker_id = i mod n
12:            Assign example x_class(i) to worker_id
13:            Assign label y_class(i) to worker_id
14:        end for
15:    end for
16: end procedure
```

Εικόνα Γ′.8: Διαμοιρασμός Διαστρωμάτωσης

C.6 Σταθεροποίηση της αρχιτεκτονική διακομιστή παραμέτρων με συστηματικό διαμοιρασμό δεδομένων πριν την εκπαίδευση

Σύμφωνα με όσα αναφέρθηκαν στην προηγούμενη ενότητα, σε αυτήν την ενότητα προτείνονται δύο τεχνικές διαμοιρασμού δεδομένων στους εργαζόμενους πριν την εκπαίδευση. Ο στόχος των τεχνικών είναι να επιτεύξουν σταθεροποίηση της διαδικασίας εκπαίδευσης, η οποία είναι ασταθής κάτω από την ασύγχρονη εκτέλεση και εξαρτάται άμεσα από το πόσο συχνά δημιουργούνται διανύσματα κλίσης με παρωχημένες παραμέτρους.

C.6.1 Προτεινόμενοι Αλγόριθμοι

Η πρώτη τεχνική που προτείνεται αφορά τον διαμοιρασμό των δεδομένων στους εργαζόμενους λαμβάνοντας υπόψιν την διαστρωμάτωση που παρέχεται από τις κλάσεις στις οποίες έχουν αντιστοιχιστεί τα δεδομένα. Ο *Διαμοιρασμός Διαστρωμάτωσης* δίνεται στην Εικόνα Γ′.8 και δεδομένου ότι κάνει προσπέλαση μία φορά σε κάθε ένα δεδομένο έχει πολυπλοκότητα $\mathcal{O}(n)$, όπου n το πλήθος του συνόλου εκπαίδευσης.

Μία δεύτερη τεχνική που προτείνεται αφορά στον εντοπισμό της κρυφής διαστρωμάτωσης που μπορεί να υπάρχει στα δεδομένα σύμφωνα με την κατανομή τους στον πολυδιάστατο χώρο. Ο αλγόριθμος κάνει προβολή των πολυδιάστατων τανυστών σε χώρο μικρότερης διάστασης μέσω του αλγορίθμου Ανάλυσης Κυριάρχων Συνιστωσών - ΑΚΣ (PCA) και εντόπιζει την κρυφή διαστρωμάτωση των δεδομένων από

```
1: procedure Διαμοιρασμος Ενημερος Κατανομης(x, y, n, classes)
2:                                  ▷ x είναι πίνακας με πολυδιαστάτους τανυστές, κάθε ένας από τους οποίους παριστάνει ένα δείγμα εκπαίδευσης
3:                                  ▷ y είναι ο αντίστοιχος πίνακας με τις κλάσεις των δειγμάτων εκπαίδευσης
4:                                  ▷ οι εργαζόμενοι απαριθμούνται με 0, 1, ..., n-1
5:                                  ▷ classes είναι το πλήθος του αλγορίθμου K-κέντρων
6:     x_flattened = flatten(x)     ▷ Επιπεδοποίηση δείγματος εκπαίδευσης
7:     x_distribution = PCA(x_flattened)
8:     cluster_ids = KMeans(x_distribution, classes)
9:     clusters = unique(cluster_ids)          ▷ Εύρεση Κρυφών Ομάδων
10:    for each cluster in clusters do
11:        x_cluster, y_cluster = in_cluster(x, y, cluster)
12:        cluster_size = length(y_cluster)
13:        if cluster_size > n then
14:            for i = 0 to cluster_size − 1 do
15:                                 ▷ Κυκλικός διαμοιρασμός κάθε κρυφής ομάδας
16:                worker_id = i mod n
17:                Assign example x_cluster(i) to worker_id
18:                Assign label y_cluster(i) to worker_id
19:            end for
20:        else
21:            Assign each x(i) in x_cluster to all workers
22:            Assign each y(i) in y_cluster to all workers
23:        end if
24:    end for
25: end procedure
```

Εικόνα Γ′.9: Διαμοιράσμος Ενήμερος Κατανομής

την κατανομή τους με χρήση του αλγορίθμου K-κέντρων. Ο αλγόριθμος *Διαμοιρασμός Ενήμερος Κατανομής* δίνεται στην Εικόνα Γ′.9. Η πολυπλοκότητα του είναι άμεσα επηρεάσμενη από την χρήση των αλγορίθμων ΑΚΣ και K-κέντρων και είναι $\mathcal{O}(n \cdot max\{d^2, k\})$, όπου n το πλήθος του συνόλου εκπαίδευσης, d το πλήθος των χαρακτηριστικών του τανυστή και k το ζητούμενο πλήθος κρυφών ομάδων που ζητείται από τον αλγόριθμο K-κέντρων.

Η χρήση του K-κέντρων μπορεί να δώσει κρυφές ομάδες που να είναι είτε πυκνές είτε αραιές όσον αφορά το πλήθος των δεδομένων που υπάρχουν σε αυτές. Θεωρώντας ως αραιές αυτές που τα σημεία δεν επαρκούν ώστε να πάρει τουλάχιστον ένα ο κάθε εργαζόμενος, ο αλγόριθμος στέλνει αυτές τις ομάδες αυτούσιες σε κάθε εργαζόμενο. Οι πυκνές γειτονιές μοιράζονται εξίσου στους εργαζόμενους.

Πίνακας Γ′.8: Σύνολα Δεδομένων

Σύνολο Δεδομένων	Μέγεθος Εικόνας	Μέγεθος Συν. Εκπαίδευσης	Μέγεθος Συν. Αξιολόγησης	Κλάσεις
CIFAR10	(36,36,3)	50,000	10,000	10
CIFAR100 (Coarse)	(36,36,3)	50,000	10,000	20
CIFAR100 (Fine)	(36,36,3)	50,000	10,000	100

Πίνακας Γ′.9: Υπερπαράμετροι εκπαίδευσης ενός κόμβου για το ResNet-56v1

Ρύθμιση	**Τιμή**
Εποχές / Βελτιστοποιητής	182 / SGD with momentum 0.9
Ρυθμός Μάθησης	Αρχικά 0.1, διαίρεση με 10 μετά από 90 και 135 εποχές
Κανονικοποίηση Συνόλου Δεδομένων	Αφαίρεση Μέσου Συνόλου Εκπαίδευσης

C.6.2 Πειραματική Διάταξη

Η πειραματική αξιολόγηση γίνεται σε μία συστοιχία 15 εικονικών μηχανών. Κάθε εικονική μηχανή έχει 4 vCPUκαι 16 GB RAM και λειτουργεί με Ubuntu 16.04.6 LTS και TensorFlow 2.3.Ως κοινό κατανεμημένο σύστημα αρχείων, χρησιμοποιείται το Apache Hadoop [69] που αναπτύσσεται με 1 namenodeκαι 14 datanodes.Στη διαδικασία εκπαίδευσης συμμετέχουν 2 διακομιστές παραμέτρων, 12 εργαζόμενοι και 1 εργασία αξιολογητή. Στη διαδικασία εκπαίδευσης, κάθε εργαζόμενος εκμεταλλεύεται μόνο δεδομένα που του έχουν εκχωρηθεί από τον αλγόριθμο διαμοιρασμού. Η πειραματική αξιολόγηση πραγματοποιείται με χρήση του δικτύου ResNet-56v1 και με τα σύνολα δεδομένων που δίνονται στον Πίνακα Γ′.8. Οι υπερπαράμετροι του δικτύου ρυθμίζονται κατάλληλα, σύμφωνα με τις προτεινόμενες για εκπαίδευση σε ένα κόμβο που δίνονται στον Πίνακα Γ′.9.

C.6.3 Πειραματική Αξιολόγηση

CIFAR10. Ο Πίνακας Γ′.10 περιγράφει τον μέσο όρο και τη διακύμανση των μετρήσεων εκπαίδευσης και αξιολόγησης για κάθε τεχνική διαμοιρασμού δεδομένων που εφαρμόζεται στο CIFAR10. Ενώ η τεχνική Διαμοιρασμού Διαστρωμάτωσης μείωνει με παρόμοιο τρόπο το σφάλμα εκπαίδευσης με τον τυχαίο διαμοιρασμό, είναι σημαντικό να παρατηρηθεί η επίδρασή της στη διακύμανση των μετρήσεων. Ο Διαμοιρασμός Διαστρωμάτωσης δίνει $1,47X$ και $3,03X$ λιγότερη διασπορά στην ακρίβεια εκπαίδευσης και αξιολόγησης σε σύγκριση με τη βασική μέθοδο διαμοιρασμού.

Ο Διαμοιρασμός Ενήμερος Κατανομής οδηγεί σε μια ελαφρά βελτίωση των μετρήσεων και μπορεί να οδηγήσει σε περαιτέρω μείωση της διασποράς, με ένα παράδειγμα στον εντοπισμό 30 κρυφών γειτονιών. Η ακρίβεια στο σύνολο εκπαίδευσης ενισχύεται κατά $0,22\%$ με $8,48X$ μικρότερη διασπορά σε σύγκριση με τον τυχαίο διαμοιρασμό. Η διακύμανση των μετρικών αξιολόγησης βελτιώνεται επίσης έως και $1,57X$.

Ο Διαμοιρασμός Ενήμερος Κατανομής παρουσιάζει μεγαλύτερη διασπορά στις με-

Πίνακας Γ′.10: Στατιστικά (5 εκτελέσεις) των τελικών τιμών Συνάρτησης Σφάλματων και Ακρίβειας Μοντελου στο Σύνολο Εκπαίδευσης και αξιολόγησης CIFAR-10 ανά τεχνική διαμοιρασμού.

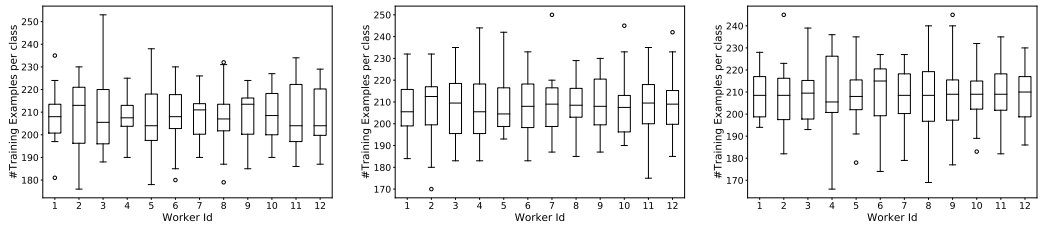
Τεχνική Διαμοιρασμού			**Μετρικές Εκπαίδευσης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			0.165786	0.001567	0.920000	0.001086
Διαστρωμάτωσης			0.165751	0.001312	**0.922397**	0.000736
Ενήμερος Κατανομής	Πλήθος Γειτονιών	20	0.165237	0.002471	0.921583	0.004059
		30	0.165166	0.001270	0.922218	**0.000128**
		40	**0.165046**	**0.000761**	0.921884	0.000851
Τεχνική Διαμοιρασμού			**Μετρικές Αξιολόγησης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			0.442370	0.006174	0.935489	0.000863
Διαστρωμάτωσης			0.444076	0.005401	0.935396	**0.000284**
Ενήμερος Κατανομής	Πλήθος Γειτονιών	20	0.445127	0.004059	0.935291	0.000648
		30	0.442075	**0.003431**	**0.935885**	0.000550
		40	**0.441692**	0.011402	0.935127	0.001358

τρικές αξιολόγησης σε σύγκριση με τον τυχαίο διαμοιρασμό μόνο εάν χρησιμοποιήσουμε μεγάλο αριθμό κρυφών γειτονιών σε σύγκριση με τον αριθμό των κλάσεων στο σύνολο δεδομένων. Αυτό οφείλεται στις αραιές γειτονιές, οι οποίες ανατίθενται αυτούσιες σε όλους τους εργαζόμενους και επομένως κάποια δείγματα εκπαίδευσης μπορεί να επαναχρησιμοποιούνται από όλους, οδηγώντας πιθανόν σε υπερπροσαρμογή του δικτύου στα δεδομένα εκπαίδευσης. Το πλήθος των αραιών γειτονιών από 3 επιλεγμένες επαναλήψεις εκπαίδευσης δίνεται στον Πίνακα Γ′.11 μαζί με την ακρίβεια στο σύνολο αξιολόγησης.

COARSE-GRAIN CIFAR-100. Ο Πίνακας Γ′.12 παρουσιάζει τις μετρικές εκπαίδευσης και αξιολόγησης για την περίπτωση χρήσης του συνόλου δεδομένων Coarse-Grain CIFAR-100. Παρατηρείται και πάλι ότι ο διαμοιρασμός διαστρωμάτωσης οδήγησε σε μοντέλο στο οποίο ελαχιστοποιούνται οι μετρήσεις εκπαίδευσης και η διασπορά των μετρήσεων αξιολόγησης. Ωστόσο, στην περίπτωση αυτή ελαχιστοποιούνται οι μετρικές αξιολόγησης όταν χρησιμοποιείται ο διαμοιρασμός Ενήμερος Κατανομής. Αυτό είναι συμβατό με όσα αναφέρθηκαν στην προηγούμενη ενότητα, με προϋπόθεση την κατάλληλη επιλογή του πλήθους κρυφών γειτονιών. Στην συγκεκριμένη περίπτωση, μία καλή επιλογή είναι οι 40 γειτονιές, δηλαδή το διπλάσιο από το αρχικό πλήθος κλάσεων του συνόλου δεδομένων. Περαιτέρω αύξηση του πλήθους

Πίνακας Γ′.11: Αραιές Γειτονιές από τον Διαμοιρασμό Ενήμερο Κατανομής (με ποικίλο πλήθος γειτονιών) με τις αντίστοιχες μετρικές αξιολόγησης για 3 επιλεγμένες εκτέλεσεις της εκπαίδευσης (CIFAR-10).Το μέσο μέγεθος γειτονιάς αναφέρεται στο μέσο πλήθος στοιχείων αραιών γειτονιών.

#Γειτονιές	Εκτέλεση#1			Εκτέλεση#2			Εκτέλεση#3		
	Αραιές Γειτονιές	Μέσο Μέγεθος	Ακρίβεια Επικ.	Αραιές Γειτονιές	Μέσο Μέγεθος	Ακρίβεια Επικ.	Αραιές Γειτονιές	Μέσο Μέγεθος	Ακρίβεια Επικ.
20	2	1	0.9352	1	1	0.9345	3	1	0.9361
30	6	2.83	0.9351	6	1.17	0.9361	4	1.25	0.9364
40	7	1.57	0.9370	8	1	0.9345	11	1.56	0.9330



(α′) Εκτέλεση #1 (β′) Εκτέλεση #2 (γ′) Εκτέλεση #3

Εικόνα Γ′.10: Θηκογράμματα αναπαράστασης του πλήθους των δεδομένων ανά κλάση στα τμήματα που ανατίθενται στους εργαζόμενους για 3 από τις εκτελέσεις.

των κρυφών γειτονιών φαίνεται να έχει χειρότερη επίδοση λόγω του φαινομένου των αραιών γειτονιών που εξηγήθηκε παραπάνω.

Έχοντας συζητήσει την ελαχιστοποίηση της διακύμανσης που μπορεί να επιτευχθεί από τον αλγόριθμο διαμοιρασμού διαστρωμάτωσης, είναι σημαντικό να κατανοήσουμε περαιτέρω γιατί ο τυχαίος διαμοιρασμός οδηγεί σε μεγαλύτερη διακύμανση στις μετρήσεις αξιολόγησης. Η Εικόνα Γ′.10 παρουσιάζει μια ομάδα θηκογραμμάτων για 3 από τις εκτελέσεις με τυχαίο διαμοιρασμό. Κάθε θηκόγραμμα περιγράφει τον πλήθος δεδομένων εκπαίδευσης από κάθε κλάση που έχει ανατεθεί σε κάθε εργαζόμενο. Το Coarse-Grain CIFAR-100 αποτελείται από 20 ισάριθμες ετικέτες. Η κατανομή του συνόλου δεδομένων σε 12 εργαζόμενους θα πρέπει να παρέχει στον καθένα περίπου 208 παραδείγματα εκπαίδευσης από κάθε κατηγορία. Οι περισσότεροι εργαζόμενοι έχουν διάμεσο αριθμό παραδειγμάτων ανά κλάση κοντά σε αυτήν την τιμή, ενώ το $(200, 215)$ είναι ένα 50%-δ.ε. στις περισσότερες περιπτώσεις. Ωστόσο, τα θηκογράμματα υποδεικνύουν ότι πολλές κλάσεις κατανέμονται άνισα στους εργαζόμενους, π.χ. στις Εικόνες Γ′.10β′ και Γ′.10γ′. Έτσι, ένας εργαζόμενος θα προσπαθήσει να προσαρμόσει το μοντέλο περισσότερο σε μια συγκεκριμένη τάξη οδηγώντας σε μεγαλύτερη απόκλιση τόσο στις μετρήσεις εκπαίδευσης όσο και στις μετρήσεις αξιολόγησης μεταξύ των επιμέρους εκτελέσεων. Φυσικά, ένας ακραίος αριθμός παραδειγμάτων ανά τάξη θα μπορούσε να βλάψει περαιτέρω τη διακύμανση, καθώς η απόκλιση του μεγέθους

Πίνακας Γ'.12: Στατιστικά (5 εκτελέσεις) των τελικών τιμών Συνάρτησης Σφάλματων και Ακρίβειας Μοντελου στο Σύνολο Εκπαίδευσης και αξιολόγησης Coarse-Grain CIFAR-100 ανά τεχνική διαμοιρασμού.

Τεχνική Διαμοιρασμού			**Μετρικές Εκπαίδευσης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			0.352169	0.004761	0.811742	0.003103
Διαστρωμάτωσης			**0.346032**	0.001903	**0.814008**	0.000766
Ενήμερος Κατανομής	Πλήθος Γειτονιών	30	0.349379	0.003350	0.812987	0.002031
		40	0.350087	**0.001738**	0.812330	**0.000535**
		60	0.347240	0.001893	0.812624	0.001138
Τεχνική Διαμοιρασμού			**Μετρικές Επικύρωσης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			1.216487	0.010065	0.805993	0.004452
Διαστρωμάτωσης			1.232794	**0.006096**	0.805841	**0.000728**
Ενήμερος Κατανομής	Πλήθος Γειτονιών	30	1.219970	0.009967	0.806072	0.000931
		40	1.211612	0.008745	**0.806237**	0.002374
		60	**1.209313**	0.021793	0.805775	0.003009

της κλάσης σε σύγκριση με τις υπόλοιπες θα δημιουργήσει περαιτέρω κυρίαρχες ή υποκυρίαρχες κλάσεις. Αυτή η μη ομοιόμορφη οπτική που έχει κάθε εργαζόμενος για τα δεδομένα στον τυχαίο διαμοιρασμό, δεν εμφανίζεται στους προτεινόμενους αλγόριθμους.

Πίνακας Γ′.13: Στατιστικά (5 εκτελέσεις) των τελικών τιμών Συνάρτησης Σφάλματων και Ακρίβειας Μοντελου στο Σύνολο Εκπαίδευσης και αξιολόγησης Fine-Grain CIFAR-100 ανά τεχνική διαμοιρασμού.

Τεχνική Διαμοιρασμού			**Μετρικές Εκπαίδευσης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			0.512164	0.009550	0.744448	0.007855
Διαστρωμάτωσης			0.516125	0.009296	0.744203	0.003107
Ενήμερος Κατανομής	Πλήθος Γειτονιών	50	0.513846	**0.004085**	0.746785	**0.000764**
		100	0.521913	0.008164	0.743808	0.005842
		200	**0.510340**	0.006247	**0.746949**	0.001134

Τεχνική Διαμοιρασμού			**Μετρικές Επικύρωσης**			
			Συνάρτηση Σφάλματος		Ακρίβεια Μοντέλου	
			Μ.Ο.	Διασπορά	Μ.Ο.	Διασπορά
Τυχαία			1.841267	0.026051	0.704707	0.002222
Διαστρωμάτωσης			**1.817435**	**0.011633**	**0.708432**	**0.001225**
Ενήμερος Κατανομής	Πλήθος Γειτονιών	50	1.835961	0.019680	0.707048	0.004526
		100	1.851682	0.015945	0.703488	0.003561
		200	1.839176	0.013421	0.708234	0.002264

FINE-GRAIN CIFAR-100 Ο πίνακας Γ′.13 αποκαλύπτει μερικά ενδιαφέροντα ευρήματα σχετικά με τις μετρήσεις επικύρωσης. Αρχικά, όπως συζητήθηκε παραπάνω, η διακύμανση των μετρικών επικύρωσης ελαχιστοποιείται όταν χρησιμοποιούνται ο διαμοιρασμός διαστρωμάτωσης. Εκτός από τη διακύμανση, ελαφρώς βελτιωμένη είναι και η εικόνα των ίδιων των μετρικών σε αυτήν την περίπτωση.

Όταν χρησιμοποιούνται διαμοιρασμός ενήμερος κατανομής, το μοντέλο φαίνεται να βελτιστοποιεί τις μετρικές εκπαίδευσης, με παρόμοια ακρίβεια επικύρωσης με αυτή της τεχνικής διαστρωμάτωσης. Αν και η τεχνική που λαμβάνει υπόψιν την κατανομή δεν ελαχιστοποιεί τη διακύμανση των μετρήσεων επικύρωσης, παρουσιάζει επίσης χαμηλότερες τιμές διακύμανσης σε σύγκριση με τη τυχαία μέθοδο.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[2] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014).

[3] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. «Speech recognition with deep recurrent neural networks». In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[4] Dario Amodei et al. «Deep speech 2: end-to-end speech recognition in English and mandarin». In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. 2016, pp. 173–182.

[5] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, 2018.

[6] Ronan Collobert and Jason Weston. «A unified architecture for natural language processing: Deep neural networks with multitask learning». In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[7] Evdokia Kassela et al. «Bigoptibase: Big data analytics for base station energy consumption optimization». In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 6098–6100.

[8] Nikodimos Provatas et al. «SELIS BDA: Big Data Analytics for the Logistics Domain». In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE. 2020, pp. 2416–2425.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Xue-Wen Chen and Xiaotong Lin. «Big data deep learning: challenges and perspectives». In: *IEEE access* 2 (2014), pp. 514–525.

[11] Chen Sun et al. «Revisiting unreasonable effectiveness of data in deep learning era». In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852.

[12] Hisham El-Amir and Mahmoud Hamdy. *Deep learning pipeline: building a deep learning model with TensorFlow*. Apress, 2019.

[13] Kaiming He et al. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[14] J. Deng et al. «ImageNet: A Large-Scale Hierarchical Image Database». In: *CVPR09*. 2009.

[15] Xiangning Chen et al. «Symbolic discovery of optimization algorithms». In: *arXiv preprint arXiv:2302.06675* (2023).

[16] Jiahui Yu et al. «Coca: Contrastive captioners are image-text foundation models». In: *arXiv preprint arXiv:2205.01917* (2022).

[17] Mitchell Wortsman et al. «Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time». In: *International Conference on Machine Learning*. PMLR. 2022, pp. 23965–23998.

[18] Xi Chen et al. «Pali: A jointly-scaled multilingual language-image model». In: *arXiv preprint arXiv:2209.06794* (2022).

[19] Zihang Dai et al. «Coatnet: Marrying convolution and attention for all data sizes». In: *Advances in neural information processing systems* 34 (2021), pp. 3965–3977.

[20] Andrew Trask, David Gilmore, and Matthew Russell. «Modeling order in neural word embeddings at scale». In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. 2015, pp. 2266–2275.

[21] Tom Brown et al. «Language models are few-shot learners». In: *arXiv preprint arXiv:2005.14165* (2020).

[22] Nestor Maslej et al. «Artificial Intelligence Index Report 2024». In: *AI Index Steering Committee, Institute for Human-Centered AI, Stanford University* (2024).

[23] Saining Xie et al. «Aggregated residual transformations for deep neural networks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[24] Jacob Devlin et al. «BERT: Pre-training of deep bidirectional transformers for language understanding». In: *arXiv preprint arXiv:1810.04805* (2018).

[25] Alec Radford et al. *Language models are unsupervised multitask learners*. OpenAI technical report. 2019.

[26] Colin Raffel et al. «Exploring the limits of transfer learning with a unified text-to-text transformer». In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67.

[27] Alexey Dosovitskiy et al. «An image is worth 16x16 words: Transformers for image recognition at scale». In: *arXiv preprint arXiv:2010.11929* (2020).

[28] Hugo Touvron et al. «LLaMA: Open and efficient foundation language models». In: *arXiv preprint arXiv:2302.13971* (2023).

[29] Ebtesam Almazrouei et al. *The Falcon Series of Open Language Models*. 2023. arXiv: 2311.16867 [cs.CL]. URL: https://arxiv.org/abs/2311.16867.

[30] Elias Jiang et al. «Mistral 7B». In: *arXiv preprint arXiv:2310.06825* (2023).

[31] OpenAI. «GPT-4 Technical Report». In: *arXiv preprint arXiv:2303.08774* (2023).

[32] Anthropic. *Claude 2*. https://www.anthropic.com/news/claude-2. 2023.

[33] Google DeepMind. *Gemini 1.5 Technical Overview*. https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/. 2024.

[34] Google DeepMind. *Gemini 2.0 Flash*. https://www.axios.com/newsletters/axios-ai-plus-cce41940-b742-11ef-b990-cdb8544335f6. 2025.

[35] Sparsh Mittal and Shraiysh Vaishay. «A survey of techniques for optimizing deep learning on GPUs». In: *Journal of Systems Architecture* 99 (2019), p. 101635.

[36] Dalton Lunga et al. «Apache spark accelerated deep learning inference for large scale satellite image analytics». In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020), pp. 271–283.

[37] Tal Ben-Nun and Torsten Hoefler. «Demystifying parallel and distributed deep learning: An in-depth concurrency analysis». In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–43.

[38] Adrián Castelló et al. «Analysis of model parallelism for distributed neural networks». In: *Proceedings of the 26th European MPI Users' Group Meeting*. 2019, pp. 1–10.

[39] Jeffrey Dean et al. «Large Scale Distributed Deep Networks». In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html.

[40] Christopher J Shallue et al. «Measuring the Effects of Data Parallelism on Neural Network Training». In: *Journal of Machine Learning Research* 20 (2019), pp. 1–49.

[41] *Measuring the Limits of Data Parallel Training for Neural Networks*. en. URL: http://ai.googleblog.com/2019/03/measuring-limits-of-data-parallel.html (visited on 09/03/2020).

[42] Yanping Huang et al. «Gpipe: Efficient training of giant neural networks using pipeline parallelism». In: *Advances in neural information processing systems* 32 (2019).

[43] Aaron Harlap et al. «Pipedream: Fast and efficient pipeline parallel dnn training». In: *arXiv preprint arXiv:1806.03377* (2018).

[44] Jeff Rasley et al. «Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters». In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 3505–3506.

[45] Qirong Ho et al. «More effective distributed ml via a stale synchronous parallel parameter server». In: *Advances in neural information processing systems* 26 (2013).

[46] Martín Abadi et al. «{TensorFlow}: A System for {Large-Scale} Machine Learning». In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[47] Tianqi Chen et al. «Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems». In: *arXiv preprint arXiv:1512.01274* (2015).

[48] Yanjun Ma et al. «PaddlePaddle: An open-source deep learning platform from industrial practice». In: *Frontiers of Data and Domputing* 1.1 (2019), pp. 105–115.

[49] Frank Seide and Amit Agarwal. «CNTK: Microsoft's open-source deep-learning toolkit». In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 2135–2135.

[50] Amazon Web Services. *Launching TensorFlow Distributed Training Easily with Horovod or Parameter Servers in Amazon SageMaker*. `https://aws.amazon.com/blogs/machine-learning/launching-tensorflow-distributed-training-easily-with-horovod-or-parameter-servers-in-amazon-sagemaker/`. AWS ML Blog post, accessed 25 May 2025. 2019.

[51] Peng Jiang et al. «A Unified Architecture for Accelerating Distributed DNN Training on GPU Clusters». In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. 2020, pp. 447–463.

[52] Xuecheng Xu et al. *MindSpore: An All-Scenario Deep Learning Computing Framework*. White Paper. Huawei Technologies Co., Ltd., 2021. URL: `https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/white_paper/MindSpore_white_paper_enV1.1.pdf`.

[53] Philipp Moritz et al. «Ray: A distributed framework for emerging {AI} applications». In: *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 2018, pp. 561–577.

[54] Amazon Web Services. *Get Started with Distributed Training in Amazon SageMaker AI*. `https://docs.aws.amazon.com/sagemaker/latest/dg/distributed-training-get-started.html`. AWS documentation, accessed 25 May 2025. 2023.

[55] Shaoqi Wang et al. «Elastic Parameter Server: Accelerating ML Training with Scalable Resource Scheduling». In: *IEEE Transactions on Parallel and Distributed Systems* 33.5 (2022), pp. 1128–1143. DOI: `10.1109/TPDS.2021.3104242`.

[56] Zhixin Huo. *Practices for Distributed Elasticity Training in the ACK Cloud-Native AI Suite*. `https://www.alibabacloud.com/blog/practices-for-distributed-elasticity-training-in-the-ack-cloud-native-ai-suite_600998`. Alibaba Cloud Community blog, accessed 25 May 2025. 2024.

[57] Alexander Smola and Shravan Narayanamurthy. «An architecture for parallel topic models». In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 703–710.

[58] Mu Li et al. «Parameter server for distributed machine learning». In: *Big learning NIPS workshop*. Vol. 6. 2. 2013.

[59] Mu Li et al. «Scaling distributed machine learning with the parameter server». In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 2014, pp. 583–598.

[60] Matei Zaharia et al. «Apache spark: a unified engine for big data processing». In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[61] Xiangrui Meng et al. «Mllib: Machine learning in apache spark». In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[62] Jason Jinquan Dai et al. «Bigdl: A distributed deep learning framework for big data». In: *Proceedings of the ACM Symposium on Cloud Computing*. 2019, pp. 50–60.

[63] Martín Abadi et al. «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems». In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX. 2016, pp. 265–283.

[64] Adam Paszke, Sam Gross, Francisco Massa, et al. «Pytorch: An imperative style, high-performance deep learning library». In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[65] David Kradofler. *Data Science Software: We screened 28,000 data job ads. Here is the software you need to know.* `https://www.datacareer.ch/blog/we-screened-28-000-data-job-ads-here-is-the-software-you-need-to-know/`. 2017.

[66] Gregory Piatetsky. *New Poll: Data Science Skills*. `https://www.kdnuggets.com/2019/08/new-poll-data-science-skills.html`. 2019.

[67] Jeffrey Dean and Sanjay Ghemawat. «MapReduce: Simplified Data Processing on Large Clusters». en. In: 2004. URL: `https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters`.

[68] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. «The Google file system». In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. SOSP '03. New York, NY, USA: Association for Computing Machinery, July 2003, 29–43. ISBN: 978-1-58113-757-6. DOI: 10.1145/945445.945450. URL: https://doi.org/10.1145/945445.945450.

[69] Apache Software Foundation. *Hadoop*. Version 0.20.2. Feb. 19, 2010. URL: https://hadoop.apache.org.

[70] Dhruba Borthakur. «HDFS Architecture Guide». en. In: (2008), p. 13.

[71] en. In: (), p. 4. URL: https://docs.aws.amazon.com/whitepapers/latest/aws-storage-services-overview/aws-storage-services-overview.pdf.

[72] Cheng-tao Chu et al. «Map-Reduce for Machine Learning on Multicore». In: *Advances in Neural Information Processing Systems*. Vol. 19. MIT Press, 2006. URL: https://proceedings.neurips.cc/paper/2006/hash/77ee3bc58ce560b86c2b59363281e914-Abstract.html.

[73] Michael Kearns. «Efficient Noise-Tolerant Learning from Statistical Queries». In: *J. ACM* 45.6 (Nov. 1998), 983–1006. ISSN: 0004-5411. DOI: 10.1145/293347.293351. URL: https://doi.org/10.1145/293347.293351.

[74] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». en. In: *Nature* 323.60886088 (Oct. 1986), 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[75] Daryl Pregibon. «Logistic Regression Diagnostics». In: *The Annals of Statistics* 9.4 (1981), pp. 705 –724. DOI: 10.1214/aos/1176345513. URL: https://doi.org/10.1214/aos/1176345513.

[76] J. A. Hartigan and M. A. Wong. «Algorithm AS 136: A K-Means Clustering Algorithm». en. In: *Applied Statistics* 28.1 (1979), p. 100. ISSN: 00359254. DOI: 10.2307/2346830.

[77] Ryan Mcdonald et al. «Efficient large-scale distributed training of conditional maximum entropy models». In: *Advances in neural information processing systems* 22 (2009).

[78] Martin Zinkevich et al. «Parallelized Stochastic Gradient Descent». In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/hash/abea47ba24142ed16b7d8fbf2c740e0d-Abstract.html.

[79] Alexandra L'Heureux et al. «Machine Learning With Big Data: Challenges and Approaches». In: *IEEE Access* 5 (2017), pp. 7776–7797.

[80] Mike Barnett et al. «Interprocessor collective communication library (InterCom)». In: *Proceedings of IEEE Scalable High Performance Computing Conference*. IEEE. 1994, pp. 357–364.

[81] Minsik Cho, Ulrich Finkler, and David Kung. «Blueconnect: Novel hierarchical all-reduce on multi-tired network for deep learning». In: *Proceedings of the 2nd SysML Conference*. 2019.

[82] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. «Optimization of collective communication operations in MPICH». In: *The International Journal of High Performance Computing Applications* 19.1 (2005), pp. 49–66.

[83] Pitch Patarasuk and Xin Yuan. «Bandwidth optimal all-reduce algorithms for clusters of workstations». In: *Journal of Parallel and Distributed Computing* 69.2 (2009), pp. 117–124.

[84] Trishul Chilimbi et al. «Project adam: Building an efficient and scalable deep learning training system». In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 2014, pp. 571–582.

[85] Eric P Xing et al. «Petuum: A new platform for distributed machine learning on big data». In: *IEEE transactions on Big Data* 1.2 (2015), pp. 49–67.

[86] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[87] *Distributed communication package - torch.distributed — PyTorch 1.6.0 documentation*. URL: `https://pytorch.org/docs/stable/distributed.html` (visited on 09/03/2020).

[88] Patrick Koch et al. «Autotune: A Derivative-free Optimization Framework for Hyperparameter Tuning». In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2018, 443–452. ISBN: 978-1-4503-5552-0. DOI: `10.1145/3219819.3219837`. URL: `https://dl.acm.org/doi/10.1145/3219819.3219837`.

[89] Suyog Gupta, Wei Zhang, and Fei Wang. «Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study». In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, Dec. 2016, 171–180. ISBN: 978-1-5090-5473-2. DOI: 10.1109/ICDM.2016.0028. URL: http://ieeexplore.ieee.org/document/7837841/.

[90] Leslie G Valiant. «A bridging model for parallel computation». In: *Communications of the ACM* 33.8 (1990), pp. 103–111.

[91] *MLlib*. https://spark.apache.org/mllib/.

[92] *Spark Broadcast Variables*. http://spark.apache.org/docs/latest/rdd-programming-guide.html#broadcast-variables.

[93] *Spark RDD*. https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#resilient-distributed-datasets-rdds.

[94] *Okeanos*. https://okeanos.grnet.gr/home/.

[95] Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris. «˜ okeanos: Building a Cloud, Cluster by Cluster». In: *IEEE internet computing* 17.3 (2013), pp. 67–71.

[96] William S Cleveland and Susan J Devlin. «Locally weighted regression: an approach to regression analysis by local fitting». In: *Journal of the American statistical association* 83.403 (1988), pp. 596–610.

[97] Daryl Pregibon. «Logistic regression diagnostics». In: *The annals of statistics* 9.4 (1981), pp. 705–724.

[98] Matt W Gardner and SR Dorling. «Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences». In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[99] *The UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml/index.php.

[100] *The HIGGS Dataset*. https://archive.ics.uci.edu/ml/datasets/HIGGS.

[101] *The YearPredictionMSD Dataset*. https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd.

[102] Yoshua Bengio. «Practical recommendations for gradient-based training of deep architectures». In: *CoRR* abs/1206.5533 (2012). URL: http://arxiv.org/abs/1206.5533.

[103] Abhay A. Ratnaparkhi, Emmanuel Pilli, and R. C. Joshi. «Survey of scaling platforms for Deep Neural Networks». In: *2016 International Conference on Emerging Trends in Communication Technologies (ETCT)*. 2016, pp. 1–6. DOI: 10.1109/ETCT.2016.7882969.

[104] Yunquan Zhang et al. «Parallel Processing Systems for Big Data: A Survey». In: *Proceedings of the IEEE* 104.11 (2016), pp. 2114–2136. DOI: 10.1109/JPROC.2016.2591592.

[105] Joost Verbraeken et al. «A Survey on Distributed Machine Learning». In: *ACM Comput. Surv.* 53.2 (Mar. 2020). ISSN: 0360-0300. DOI: 10.1145/3377454. URL: https://doi.org/10.1145/3377454.

[106] Shaohuai Shi et al. «A Quantitative Survey of Communication Optimizations in Distributed Deep Learning». In: *IEEE Network* 35.3 (2021), pp. 230–237. DOI: 10.1109/MNET.011.2000530.

[107] Shuo Ouyang et al. «Communication optimization strategies for distributed deep neural network training: A survey». In: *Journal of Parallel and Distributed Computing* 149 (2021), pp. 52–65.

[108] Barbara Kitchenham et al. «Systematic literature reviews in software engineering – A systematic literature review». In: *Information and Software Technology* 51.1 (2009). Special Section - Most Cited Articles in 2002 and Regular Research Papers, pp. 7–15. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2008.09.009. URL: https://www.sciencedirect.com/science/article/pii/S0950584908001390.

[109] Benjamin Recht et al. «Hogwild: A lock-free approach to parallelizing stochastic gradient descent». In: *Advances in neural information processing systems*. 2011, 693–701.

[110] Guanhua Wang et al. «ZeRO++: Extremely Efficient Collective Communication for Giant Model Training». In: *ICLR 2024*. May 2024. URL: https://www.microsoft.com/en-us/research/publication/zero-extremely-efficient-collective-communication-for-giant-model-training/.

[111] Samyam Rajbhandari et al. «Zero: Memory optimizations toward training trillion parameter models». In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–16.

[112] Alexander Sergeev and Mike Del Balso. «Horovod: fast and easy distributed deep learning in TensorFlow». In: *arXiv preprint arXiv:1802.05799* (2018).

[113] Xiangru Lian et al. «Asynchronous parallel stochastic gradient for nonconvex optimization». In: *Advances in neural information processing systems* 28 (2015).

[114] Aaron Harlap et al. «Addressing the straggler problem for iterative convergent parallel ML». In: *Proceedings of the seventh ACM symposium on cloud computing*. 2016, pp. 98–111.

[115] Hao Zhang et al. «Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters». In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017, pp. 181–193.

[116] Qihua Zhou et al. «Falcon: Towards computation-parallel deep learning in heterogeneous parameter server». In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 196–206.

[117] Qihua Zhou et al. «Falcon: Addressing stragglers in heterogeneous parameter server via multiple parallelism». In: *IEEE Transactions on Computers* 70.1 (2020), pp. 139–155.

[118] Shuai Zheng, Ziyue Huang, and James Kwok. «Communication-efficient distributed blockwise momentum SGD with error-feedback». In: *Advances in Neural Information Processing Systems* 32 (2019).

[119] Hanlin Tang et al. «Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression». In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6155–6165.

[120] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. «Tictac: Accelerating distributed deep learning with communication scheduling». In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 418–430.

[121] Chia-Yu Chen et al. «Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training». In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13551–13563.

[122] Yanghua Peng et al. «Optimus: an efficient dynamic resource scheduler for deep learning clusters». In: *Proceedings of the Thirteenth EuroSys Conference*. 2018, pp. 1–14.

[123] Yuzhen Huang et al. «Flexps: Flexible parallelism control in parameter server architecture». In: *Proceedings of the VLDB Endowment* 11.5 (2018), pp. 566–579.

[124] Liang Luo et al. «Parameter hub: a rack-scale parameter server for distributed deep neural network training». In: *Proceedings of the ACM Symposium on Cloud Computing*. 2018, pp. 41–54.

[125] Alexander Renz-Wieland et al. «Dynamic parameter allocation in parameter servers». In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 1877–1890.

[126] Alexander Renz-Wieland et al. «Just move it! dynamic parameter allocation in action». In: *Proceedings of the VLDB Endowment* 14.12 (2021), pp. 2707–2710.

[127] Alexander Renz-Wieland et al. «NuPS: A Parameter Server for Machine Learning with Non-Uniform Parameter Access». In: *Proceedings of the 2022 International Conference on Management of Data*. 2022, pp. 481–495.

[128] Mu Li et al. «Communication efficient distributed machine learning with the parameter server». In: *Advances in Neural Information Processing Systems* 27 (2014).

[129] Jinliang Wei et al. «Managed communication and consistency for fast data-parallel iterative analytics». In: *Proceedings of the Sixth ACM Symposium on Cloud Computing*. 2015, pp. 381–394.

[130] Jun Zhou et al. «Kunpeng: Parameter server based distributed learning systems and its applications in alibaba and ant financial». In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 1693–1702.

[131] Jun Zhou et al. «PSMART: parameter server based multiple additive regression trees system». In: *Proceedings of the 26th International Conference on World Wide Web Companion*. 2017, pp. 879–880.

[132] Po-Yen Wu, Pangfeng Liu, and Jan-Jan Wu. «Versatile Communication Optimization for Deep Learning by Modularized Parameter Server». In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 366–371.

[133] Wei Zhang et al. «Staleness-aware async-sgd for distributed deep learning». In: *arXiv preprint arXiv:1511.05950* (2015).

[134] Sanghamitra Dutta et al. «Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD». In: *International conference on artificial intelligence and statistics*. PMLR. 2018, pp. 803–812.

[135] Aurick Qiao et al. «Litz: Elastic Framework for {High-Performance} Distributed Machine Learning». In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, pp. 631–644.

[136] Haozhao Wang, Song Guo, and Ruixuan Li. «OSP: Overlapping computation and communication in parameter server for fast machine learning». In: *Proceedings of the 48th International Conference on Parallel Processing*. 2019, pp. 1–10.

[137] Xing Zhao et al. «Dynamic stale synchronous parallel distributed training for deep learning». In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 1507–1517.

[138] Raajay Viswanathan, Arjun Balasubramanian, and Aditya Akella. «Network-accelerated distributed machine learning for multi-tenant settings». In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 447–461.

[139] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. «Sparsified SGD with memory». In: *Advances in neural information processing systems* 31 (2018).

[140] Wei Wen et al. «Terngrad: Ternary gradients to reduce communication in distributed deep learning». In: *Advances in neural information processing systems* 30 (2017).

[141] Dan Alistarh et al. «QSGD: Communication-efficient SGD via gradient quantization and encoding». In: *Advances in neural information processing systems* 30 (2017).

[142] Hongyi Wang et al. «Atomo: Communication-efficient learning via atomic sparsification». In: *Advances in Neural Information Processing Systems* 31 (2018).

[143] Guoxin Cui et al. «Mqgrad: Reinforcement learning of gradient quantization in parameter server». In: *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*. 2018, pp. 83–90.

[144] Foivos Alimisis, Peter Davies, and Dan Alistarh. «Communication-efficient distributed optimization with quantized preconditioners». In: *International Conference on Machine Learning*. PMLR. 2021, pp. 196–206.

[145] Nikita Ivkin et al. «Communication-efficient distributed SGD with sketching». In: *Advances in Neural Information Processing Systems* 32 (2019).

[146] Lusine Abrahamyan et al. «Learned gradient compression for distributed deep learning». In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[147] Hang Xu et al. «Grace: A compressed communication framework for distributed machine learning». In: *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*. IEEE. 2021, pp. 561–572.

[148] Anand Jayarajan et al. «Priority-based parameter propagation for distributed DNN training». In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 132–145.

[149] Shaoqi Wang et al. «Overlapping communication with computation in parameter server for scalable dl training». In: *IEEE Transactions on Parallel and Distributed Systems* 32.9 (2021), pp. 2144–2159.

[150] Indu Thangakrishnan et al. «Herring: Rethinking the parameter server at scale for the cloud». In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–13.

[151] Lele Yut et al. «LDA* a robust and large-scale topic modeling system». In: *Proceedings of the VLDB Endowment* 10.11 (2017), pp. 1406–1417.

[152] Heng Pan et al. «Dissecting the communication latency in distributed deep sparse learning». In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 528–534.

[153] Yangrui Chen et al. «Elastic parameter server load distribution in deep learning clusters». In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 507–521.

[154] Ganesh Ananthanarayanan et al. «Effective Straggler Mitigation: Attack of the Clones». In: *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX Association, Apr. 2013, pp. 185–198. ISBN: 978-1-931971-00-3. URL: `https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/ananthanarayanan`.

[155] Chen Chen et al. «Semi-dynamic load balancing: Efficient distributed learning in non-dedicated environments». In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 431–446.

[156] John Langford, Alexander J. Smola, and Martin Zinkevich. «Slow Learners Are Fast». In: *Proceedings of the 22nd International Conference on Neural Information Processing Systems*. NIPS'09. Vancouver, British Columbia, Canada: Curran Associates Inc., 2009, 2331–2339. ISBN: 9781615679119.

[157] Jiawei Jiang et al. «Heterogeneity-aware distributed parameter servers». In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 463–478.

[158] Huihuang Yu et al. «Accelerating distributed training in heterogeneous clusters via a straggler-aware parameter server». In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE. 2019, pp. 200–207.

[159] Tyler Akidau et al. «Millwheel: Fault-tolerant stream processing at internet scale». In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1033–1044.

[160] Tolga Acar et al. «Key management in distributed systems». In: *Microsoft Research* (2010), pp. 1–14.

[161] Miguel Correia et al. «Practical Hardening of {Crash-Tolerant} Systems». In: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 2012, pp. 453–466.

[162] Zhaoning Zhang et al. «A Quick Survey on Large Scale Distributed Deep Learning Systems». In: *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 2018, pp. 1052–1056. DOI: 10.1109/PADSW.2018.8644613.

[163] Brendan McMahan et al. «Communication-efficient learning of deep networks from decentralized data». In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[164] Jakub Konečnỳ et al. «Federated learning: Strategies for improving communication efficiency». In: *arXiv preprint arXiv:1610.05492* (2016).

[165] Zhixiong Xu et al. «Meta weight learning via model-agnostic meta-learning». In: *Neurocomputing* 432 (2021), pp. 124–132.

[166] Yuanhao Xiong et al. «Feddm: Iterative distribution matching for communication-efficient federated learning». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16323–16332.

[167] Keith Bonawitz et al. «Towards federated learning at scale: System design». In: *Proceedings of machine learning and systems* 1 (2019), pp. 374–388.

[168] Sunwoo Lee, Tuo Zhang, and A Salman Avestimehr. «Layer-wise adaptive model aggregation for scalable federated learning». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 2023, pp. 8491–8499.

[169] Song Han et al. «Practical and Robust Federated Learning With Highly Scalable Regression Training». In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[170] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. «Personalized federated learning: A meta-learning approach». In: *arXiv preprint arXiv:2002.07948* (2020).

[171] Rui Ye et al. «Personalized federated learning with inferred collaboration graphs». In: *International Conference on Machine Learning*. PMLR. 2023, pp. 39801–39817.

[172] Fu-En Yang, Chien-Yi Wang, and Yu-Chiang Frank Wang. «Efficient model personalization in federated learning via client-specific prompt generation». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 19159–19168.

[173] Aleksander Madry et al. «Towards Deep Learning Models Resistant to Adversarial Attacks». In: *International Conference on Learning Representations*. 2018.

[174] Manisha Guduri et al. «Blockchain-based Federated Learning Technique for Privacy Preservation and Security of Smart Electronic Health Records». In: *IEEE Transactions on Consumer Electronics* (2023).

[175] Truc Nguyen and My T Thai. «Preserving privacy and security in federated learning». In: *IEEE/ACM Transactions on Networking* (2023).

[176] Chen Zhang et al. «A survey on federated learning». In: *Knowledge-Based Systems* 216 (2021), p. 106775.

[177] Wei Yang Bryan Lim et al. «Federated learning in mobile edge networks: A comprehensive survey». In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 2031–2063.

[178] Dinh C Nguyen et al. «Federated learning for internet of things: A comprehensive survey». In: *IEEE Communications Surveys & Tutorials* 23.3 (2021), pp. 1622–1658.

[179] Qinbin Li et al. «A survey on federated learning systems: Vision, hype and reality for data privacy and protection». In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[180] Ji Liu et al. «From distributed machine learning to federated learning: A survey». In: *Knowledge and Information Systems* 64.4 (2022), pp. 885–917.

[181] Xuefei Yin, Yanming Zhu, and Jiankun Hu. «A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions». In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–36.

[182] Viraaji Mothukuri et al. «A survey on security and privacy of federated learning». In: *Future Generation Computer Systems* 115 (2021), pp. 619–640.

[183] Dinh C Nguyen et al. «Federated learning for smart healthcare: A survey». In: *ACM Computing Surveys (CSUR)* 55.3 (2022), pp. 1–37.

[184] Alysa Ziying Tan et al. «Towards personalized federated learning». In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[185] Youyang Qu et al. «Blockchain-enabled federated learning: A survey». In: *ACM Computing Surveys* 55.4 (2022), pp. 1–35.

[186] Jie Wen et al. «A survey on federated learning: challenges and applications». In: *International Journal of Machine Learning and Cybernetics* 14.2 (2023), pp. 513–535.

[187] Mang Ye et al. «Heterogeneous federated learning: State-of-the-art and research challenges». In: *ACM Computing Surveys* 56.3 (2023), pp. 1–44.

[188] Y Supriya and Thippa Reddy Gadekallu. «A Survey on Soft Computing Techniques for Federated Learning-Applications, Challenges and Future Directions». In: *ACM Journal of Data and Information Quality* (2023).

[189] Shijian Li et al. «Sync-switch: Hybrid parameter synchronization for distributed deep learning». In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2021, pp. 528–538.

[190] Qinyi Luo et al. «Prague: High-performance heterogeneity-aware asynchronous decentralized training». In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 401–416.

[191] Xiangru Lian et al. «Asynchronous decentralized parallel stochastic gradient descent». In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3043–3052.

[192] Guangmeng Zhou et al. «FedPAGE: Pruning Adaptively Toward Global Efficiency of Heterogeneous Federated Learning». In: *IEEE/ACM Transactions on Networking* 32.3 (2024), pp. 1873–1887. DOI: 10.1109/TNET.2023.3328632.

[193] Canh T. Dinh et al. «Federated Learning Over Wireless Networks: Convergence Analysis and Resource Allocation». In: *IEEE/ACM Transactions on Networking* 29.1 (2021), pp. 398–409. DOI: 10.1109/TNET.2020.3035770.

[194] Jianfeng Lu et al. «Toward Personalized Federated Learning Via Group Collaboration in IIoT». In: *IEEE Transactions on Industrial Informatics* 19.8 (2023), pp. 8923–8932. DOI: 10.1109/TII.2022.3223234.

[195] Yuang Jiang et al. «Model Pruning Enables Efficient Federated Learning on Edge Devices». In: *IEEE Transactions on Neural Networks and Learning Systems* 34.12 (2023), pp. 10374–10386. DOI: 10.1109/TNNLS.2022.3166101.

[196] G. Damaskinos et al. «Asynchronous Byzantine Machine Learning (The Case of SGD)». In: *Proceedings of the International Conference on Machine Learning*. PMLR, 2018, pp. 1145–1154.

[197] R. Viswanathan, A. Balasubramanian, and A. Akella. «Network-Accelerated Distributed Machine Learning for Multi-Tenant Settings». In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 447–461.

[198] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. «On the stratification of multi-label data». In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 145–158.

[199] L. Oakden-Rayner et al. «Hidden Stratification Causes Clinically Meaningful Failures in Machine Learning for Medical Imaging». In: *Proceedings of the ACM Conference on Health, Inference, and Learning*. 2020, pp. 151–159.

[200] Joel Nishimura and Johan Ugander. «Restreaming graph partitioning: simple versatile algorithms for advanced balancing». In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 1106–1114.

[201] Jinyang Gao et al. «DSH: data sensitive hashing for high-dimensional k-nnsearch». In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 1127–1138.

[202] N. Polyzotis et al. «Data Lifecycle Challenges in Production Machine Learning: A Survey». In: *ACM SIGMOD Record* 47.2 (2018), pp. 17–28.

[203] K. Hsieh et al. «The Non-IID Data Quagmire of Decentralized Machine Learning». In: *Proceedings of the International Conference on Machine Learning*. PMLR, 2020, pp. 4387–4398.

[204] Zeng Miao et al. «Heterogeneity-aware All-Reduce for Distributed Deep Learning». In: *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. 2021, pp. 1–16.

[205] Suyog Gupta, Wei Zhang, and Fei Wang. «Model accuracy and runtime tradeoff in distributed deep learning: A systematic study». In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. 2016, pp. 171–180.

[206] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[207] Yuan Lu, Hanrui Wang, Yubo Mao, et al. «LoTUS: Low-latency Transfer for Ultra-scale Distributed Deep Learning». In: *2020 USENIX Annual Technical Conference (USENIX ATC)*. 2020, pp. 275–292.

[208] Brendan McMahan et al. «Communication-Efficient Learning of Deep Networks from Decentralized Data». In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2017, pp. 1273–1282.

[209] Martin Jaggi, Sebastian Li, Maximilian Hurtig, et al. «On the Communication Complexity of Distributed Convex Learning and Optimization». In: *Advances in Neural Information Processing Systems*. 2014, pp. 1810–1818.

[210] Shaohuai Shi et al. «A Quantitative Survey of Communication Optimizations in Distributed Deep Learning». In: *IEEE Network* 35.3 (2020), pp. 230–237.

[211] Nikodimos Provatas. «Exploiting data distribution in distributed learning of deep classification models under the parameter server architecture.» In: *Proceedings of the VLDB 2021 PhD Workshop (VLDB-PhD 2021), Copenhagen, Denmark*. Vol. 2971. 2021.

[212] Nikodimos Provatas, Ioannis Konstantinou, and Nectarios Koziris. «Towards faster distributed deep learning using data hashing techniques». In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 6189–6191.

[213] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Commun. ACM* 60.6 (2017), 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[214] Friedrich-Wilhelm Wellmer. «The Normal Distribution». In: *Statistical Evaluations in Exploration for Mineral Deposits*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 27–30. ISBN: 978-3-642-60262-7. DOI: 10.1007/978-3-642-60262-7_4. URL: https://doi.org/10.1007/978-3-642-60262-7_4.

[215] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[216] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[217] Srikanth Kandula et al. «Experiences with approximating queries in Microsoft's production big-data clusters». In: *Proceedings of the VLDB Endowment* 12.12 (2019), pp. 2131–2142.

[218] Jose Picado, Arash Termehchy, and Sudhanshu Pathak. «Learning efficiently over heterogeneous databases». In: *Proceedings of the VLDB Endowment* 11.12 (2018), pp. 2066–2069.

[219] Luke Oakden-Rayner et al. «Hidden stratification causes clinically meaningful failures in machine learning for medical imaging». In: *Proceedings of the ACM conference on health, inference, and learning*. 2020, pp. 151–159.

[220] Nimit Sohoni et al. «No subclass left behind: Fine-grained robustness in coarse-grained classification problems». In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19339–19352.

[221] Erich Schubert et al. «A framework for clustering uncertain data». In: *Proceedings of the VLDB Endowment* 8.12 (2015), pp. 1976–1979.

[222] *Clustering*. URL: https://spark.apache.org/docs/3.1.2/ml-clustering.html#k-means.

[223] Alessandro Lulli et al. «NG-DBSCAN: scalable density-based clustering for arbitrary data». In: *Proceedings of the VLDB Endowment* 10.3 (2016), pp. 157–168.

[224] Yoshua Bengio et al. «Curriculum learning». In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[225] Ilya Loshchilov and Frank Hutter. «Online batch selection for faster training of neural networks». In: *arXiv preprint arXiv:1511.06343* (2015).

[226] Li Deng. «The mnist database of handwritten digit images for machine learning research». In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[227] KJ Joseph, Krishnakant Singh, and Vineeth N Balasubramanian. «Submodular batch selection for training deep neural networks». In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2019, pp. 2677–2683.

[228] Tianyi Zhou, Shengjie Wang, and Jeffrey Bilmes. «Curriculum learning by dynamic instance hardness». In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 8602–8613.

[229] Hwanjun Song et al. «Carpe Diem, Seize the Samples Uncertain" at the Moment" for Adaptive Batch Selection». In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 1385–1394.

[230] Md Maniruzzaman et al. «Accurate diabetes risk stratification using machine learning: role of missing value and outliers». In: *Journal of medical systems* 42.5 (2018), p. 92.

[231] URL: `https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shard`.

[232] Stuart Lloyd. «Least squares quantization in PCM». In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[233] James MacQueen et al. «Some methods for classification and analysis of multivariate observations». In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[234] Karl Pearson F.R.S. «On lines and planes of closest fit to systems of points in space». In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[235] *sklearn.decomposition.IncrementalPCA¶*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html.

[236] *sklearn.cluster.KMeans¶*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[237] S. Shi et al. «Benchmarking State-of-the-Art Deep Learning Software Tools». In: *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. 2016, pp. 99–104.

[238] *Dimensionality Reduction - RDD-based API*. URL: https://spark.apache.org/docs/latest/mllib-dimensionality-reduction.html#principal-component-analysis-pca.

[239] Deepak Narayanan et al. «Efficient large-scale language model training on GPU clusters using megatron-LM». In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: 10.1145/3458817.3476209. URL: https://doi.org/10.1145/3458817.3476209.

[240] Ziheng Jiang et al. «MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs». In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 745–760. ISBN: 978-1-939133-39-7. URL: https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng.

[241] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL]. URL: https://arxiv.org/abs/1606.05250.

[242] Alex Wang et al. «Superglue: A stickier benchmark for general-purpose language understanding systems». In: *Advances in neural information processing systems* 32 (2019).

[243] Percy Liang et al. «Holistic Evaluation of Language Models». In: *Transactions on Machine Learning Research* (2023). Featured Certification, Expert Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=iO4LZibEqW.

[244] Paul Pu Liang et al. *HEMM: Holistic Evaluation of Multimodal Foundation Models*. 2024. arXiv: 2407.03418 [cs.LG]. URL: https://arxiv.org/abs/2407.03418.

[245] Dan Hendrycks et al. «Measuring Massive Multitask Language Understanding». In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=d7KBjmI3GmQ.

[246] Wei-Lin Chiang et al. «Chatbot arena: an open platform for evaluating LLMs by human preference». In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2025.

[247] Stephanie Lin et al. «TruthfulQA: Measuring How Models Mimic Human Falsehoods». In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3214–3252. DOI: 10.18653/v1/2022.acl-long.229. URL: https://aclanthology.org/2022.acl-long.229/.

[248] Junyi Li et al. «HaluEval: A Large-Scale Hallucination Evaluation Benchmark for Large Language Models». In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6449–6464. DOI: 10.18653/v1/2023.emnlp-main.397. URL: https://aclanthology.org/2023.emnlp-main.397/.

[249] Xiang Yue et al. «Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 9556–9567.

[250] Kanishk Gandhi et al. «Understanding social reasoning in language models with language models». In: *Advances in Neural Information Processing Systems* 36 (2024).

[251] Xiao Liu et al. «AgentBench: Evaluating LLMs as Agents». In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=zAdUB0aCTQ.

[252] Jae-Won Chung et al. «Reducing Energy Bloat in Large Model Training». In: *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. SOSP '24. New York, NY, USA: Association for Computing Machinery, 2024, 144–159. ISBN: 9798400712517. DOI:

10.1145/3694715.3695970. URL: https://doi.org/10.1145/3694715.3695970.

[253] Mauricio Fadel Argerich and Marta Patiño-Martínez. «Measuring and Improving the Energy Efficiency of Large Language Models Inference». In: *IEEE Access* 12 (2024), pp. 80194–80207. DOI: 10.1109/ACCESS.2024.3409745.

[254] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. «Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training». In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 119–139. ISBN: 978-1-939133-33-5. URL: https://www.usenix.org/conference/nsdi23/presentation/you.

[255] Sangjin Choi et al. «EnvPipe: Performance-preserving DNN Training Framework for Saving Energy». In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, July 2023, pp. 851–864. ISBN: 978-1-939133-35-9. URL: https://www.usenix.org/conference/atc23/presentation/choi.

[256] Sebastian Ruder. «An overview of gradient descent optimization algorithms». In: *arXiv preprint arXiv:1609.04747* (2016).

[257] Martin Zinkevich et al. «Parallelized stochastic gradient descent». In: *Advances in neural information processing systems*. 2010, 2595–2603.

[258] Diederik P. Kingma and Jimmy Ba. «Adam: A method for stochastic optimization». In: *arXiv preprint arXiv:1412.6980* (2014).

[259] Yoshua Bengio. «Rmsprop and equilibrated adaptive learning rates for nonconvex optimization». In: *corr abs/1502.04390* (2015).

[260] Ilya Sutskever et al. «On the importance of initialization and momentum in deep learning». In: *International conference on machine learning*. 2013, 1139–1147.

[261] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. «Similarity search in high dimensions via hashing». In: *Vldb*. Vol. 99. 6. 1999, pp. 518–529.

[262] Piotr Indyk and Rajeev Motwani. «Approximate nearest neighbors: towards removing the curse of dimensionality». In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM. 1998, pp. 604–613.

[263] Jure Leskovec, Anand Rajaraman, and Jeff Ullman. «Mining of Massive Datasets». In: Cambridge University Press, Oct. 2011. Chap. 3, pp. 80–84.

[264] Matei Zaharia et al. «Spark: Cluster Computing with Working Sets». In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010.