Comparison between Python scaling frameworks for big data analysis and ML

Γεωργία Μπουσμπουκέα - el19059
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων

Γεώργιος Μπαρής – el19866
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων

Περίληψη:

Το παρόν κείμενο αποτελεί την εξαμηνιαία εργασία για το μάθημα “Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων” για το έτος 2023-2024. Ο τίτλος της εργασίας είναι “Comparison between Python scaling frameworks for big data analysis and ML”, με ειδική εστίαση στα εργαλεία Ray και PyTorch. Σκοπός της εργασίας είναι να προσφέρει μια σφαιρική σύγκριση ανάμεσα στα δύο λογισμικά, εστιάζοντας στην εφαρμογή τους σε περιβάλλοντα Μηχανικής Μάθησης και ανάλυσης μεγάλων όγκων δεδομένων.

Η διαδικασία ανάπτυξης της εργασίας περιλαμβάνει τη συγγραφή κώδικα σε Python, με ενσωμάτωση των προαναφερθέντων εργαλείων. Τα σενάρια που αναπτύξαμε περιλαμβάνουν εφαρμογές Binary Classification σε δεδομένα CSV, Personalised Page Rank σε γράφους και Classification σε image δεδομένα. Η αξιολόγηση της απόδοσης των εργαλείων βασίζεται σε παραμέτρους όπως το μέγεθος της εισόδου και ο αριθμός των executors στο cluster.

Στο πλαίσιο των μετρικών απόδοσης, εξετάζουμε τον χρόνο εκτέλεσης, την κλιμάκωση και την ευκολία χρήσης. Η πρώτη μετρική παρέχει πληροφορίες για την αποτελεσματικότητα της εκτέλεσης των εφαρμογών, ενώ η δεύτερη αξιολογεί την ικανότητα των εργαλείων να επεκτείνονται σε cluster με πολλούς κόμβους. Το πλαίσιο της ευκολίας χρήσης εστιάζει στη διαδικασία εγκατάστασης, διαμόρφωσης και ανάπτυξης του κώδικα και της οπτικοποίησης του συστήματος. Η σαφήνεια της διαδικασίας είναι κρίσιμη για την αποδοτική χρήση των εργαλείων.

Μέσω αυτής της εργασίας, στοχεύουμε στην κατανόηση των δυνατοτήτων και των περιορισμών των εργαλείων Ray και PyTorch στον τομέα της Μηχανικής Μάθησης και της ανάλυσης μεγάλων όγκων δεδομένων.

Λέξεις κλειδιά: Ray, Pytorch, Binary Classification, Personalised Page Rank

I. Περιγραφή του set-up του συστήματος

A. Cluster

Για τους σκοπούς της εργασίας χρησιμοποιήσαμε εικονικά μηχανήματα από τον ‘Oceanos’. Αρχικά, συνδέουμε τα μηχανήματά μας σε ένα private ipv4 δίκτυο και φροντίζουμε την απρόσκοπτη πρόσβαση μεταξύ τους μέσω ssh με χρήση δημοσίων κλειδιών, απαραίτητο για την κατανομή των διεργασιών κατά την εκτέλεση του κώδικα. Έπειτα, αναβαθμίσαμε το λογισμικό σε Ubuntu 22.04, για να υπάρχει δυνατότητα εγκατάστασης των τελευταίων εκδόσεων βιβλιοθηκών και πακέτων που θα χρειαστούμε. Λαμβάνοντας υπόψιν ότι τα δεδομένα που θα χρησιμοποιήσουμε είναι της τάξεως των GB, θα διαμορφώσουμε κατάλληλα το HDFS, ώστε να γίνει καλύτερα η κατανομή τους μεταξύ των κόμβων. Το cluster μας αποτελείται από δύο κόμβους: okeanos-master, okeanos-server2, καθένας από τους οποίους διαθέτει 4 CPUs, 30GB δίσκο και 8GB RAM.

B. Pytorch

Η εγκατάσταση του Pytorch είναι σχετικά απλή. Πλοηγούμαστε στην επίσημη ιστοσελίδα και, με δεδομένες τις παραμέτρους τους συστήματός μας (τα εικονικά μηχανήματα έχουν λειτουργικό linux και δεν διαθέτουν GPU), θα εγκαταστήσουμε το Pytorch με χρήση του pip:

pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu

Μπορούμε να έχουμε πρόσβαση στο pytorch απλά συμπεριλαμβάνοντας “import torch” στον κώδικά μας.

C. Ray

Η διαμόρφωση του Ray είναι ελαφρώς πιο σύνθετη. Αρχικά θα εγκαταστήσουμε το πακέτο με τις απαραίτητες βιβλιοθήκες:

pip install ray[default] pip install ray[train]

Για να έχουμε πρόσβαση στο Ray θα πρέπει να εκκινήσουμε το ray cluster με την εντολή: “ray start –head" στον master κόμβο και να προσθέσουμε τον worker με: “ray start --address=<private IP of master node>:6379” (αποδίδεται αυτόματα η θύρα 6379). Ακόμη, θα πρέπει να χρησιμοποιήσουμε το “ray.init()” στην αρχή του προγράμματός μας, προκειμένου να συνδεθεί με το ray cluster.
Το Ray παρέχει ακόμη το ‘Ray Dashboard’, ένα γραφικό περιβάλλον, όπου μπορούμε να παρακολουθούμε την κατάσταση του cluster, των πόρων και την εκτέλεση των διεργασιών. Αυτόματα του αποδίδεται η θύρα 8265 του localhost. Καθώς στην περίπτωσή μας επρόκειτο για εικονικά μηχανήματα, για τα οποία δεν υπάρχει πρόσβαση σε web browser, θα δημιουργήσουμε ένα tunnel για port forwarding μεταξύ του τοπικού Η/Υ και του master κόμβου με την εντολή:

ssh -N -L 8265:localhost:8265
<username>@<public IP>

Πλέον μπορούμε να παρακολουθούμε το Dashboard στην διεύθυνση http://127.0.0.1:8265.

II. Περιγραφή του λογισμικού

Τα Ray, Pytorch είναι open-source βιβλιοθήκες της Python, που σχεδιάστηκαν για να επιτελούν διαφορετικούς σκοπούς στον ευρύτερο τομέα της Μηχανικής Μάθησης και των Κατανεμημένων Συστημάτων.

Συγκεκριμένα, το Pytorch παρέχει ένα API με εκτενή γκάμα εργαλείων για τη δημιουργία, την εκπαίδευση, και τον έλεγχο νευρωνικών δικτύων. Κάνοντας wrap το μοντέλο του νευρωνικού που θα χρησιμοποιήσουμε με την κλάση DistributedDataParallel, ορίζουμε να κατανεμηθεί το training στις διεργασίες. Διαθέτει επίσης τις κλάσεις ‘Dataset’, ‘Dataloader’, ‘Distributed Sampler’, οι οποίες απαλλάσσουν τον χρήστη από την διαχείριση της μνήμης σε περίπτωση χρήσης dataset που την υπερβαίνει. Ειδικότερα ένα στιγμιότυπο της κλάσης Dataloader παίρνει ως είσοδο ένα στιγμιότυπο της Dataset και δημιουργεί ένα “επαναληπτικό” αντικείμενο με χρήση batches. Οπότε πλέον το dataset φορτώνεται και υπόκειται σε επεξεργασία ανά batch, χωρίς να χρειάζεται να χωράει όλο στην μνήμη. Όσον αφορά στο κομμάτι της διανομής των διεργασιών στο Pytorch, υπάρχει η υποβιβλιοθήκη torch.distributed, μέσω της οποίας ορίζεται ένα group διεργασιών που θα τρέξουν παράλληλα σε batches (distributed.init_process_group()) και γίνεται η τελική συλλογή των αποτελεσμάτων (distributed.all_gather()).

Το Ray, από την άλλη, κάνει την κατανομή των διεργασιών αυτόματα, ανάλογα με τους διαθέσιμους πόρους στους κόμβους του cluster, αρκεί να αρχικοποιηθεί (ray.init()) και να δηλωθούν οι συναρτήσεις που θέλουμε να εκτελεστούν παράλληλα (@ray.remote()), διατηρώντας συγχρόνως την πρόσβαση στις προαναφερθείσες κλάσεις του Pytorch.

III. Περιγραφή των πειραμάτων

A. Binary Classification σε csv data

Το πρώτο πείραμα που επιλέξαμε να υλοποιήσουμε είναι στον τομέα της Μηχανικής Μάθησης και είναι Δυαδική Ταξινόμηση σε csv αρχείο. Προμηθευτήκαμε τα δεδομένα μας από το kaggle [1]. Οι στήλες των δεδομένων έχουν να κάνουν με διάφορα χαρακτηριστικά ουράνιων σωμάτων, π.χ. ταχύτητα σε κάθε άξονα, Petrosian και exponential radius υπό διάφορα φίλτρα, γαλαξιακό μήκος και πλάτος, και η προβλεπόμενη στήλη είναι η ‘type’, που δηλώνει αν το δείγμα πρόκειται για ‘star’ ή για ‘galaxy’. Στα δεδομένα αυτά θα εφαρμόσουμε EDA μέσω python, οπότε κάνουμε scaling, αφαιρούμε τα χαρακτηριστικά με υψηλό βαθμό συσχέτισης με άλλα και αντιστοιχούμε την τιμή ‘galaxy’ σε ‘1’ και ‘star’ σε ‘0’, ώστε να γίνει εύκολα η ταξινόμηση.

Pytorch: Ορίζουμε το Dataset έτσι ώστε να έχει ως αντικείμενα τα χαρακτηριστικά και τα αντίστοιχα labels μέσω της συνάρτησης __getitem__(). Για την ταξινόμηση θα χρησιμοποιήσουμε ένα απλό νευρωνικό δίκτυο, χωρίς κρυφά επίπεδα, που θα παίρνει ως είσοδο τα χαρακτηριστικά του δείγματος και θα δίνει, μέσω σιγμοειδούς συνάρτησης ενεργοποίησης, μία έξοδο, την πιθανότητα το δείγμα να ανήκει στην κλάση '1'.
Αρχικά ορίζουμε το process_group, το μοντέλο μας και τα χαρακτηριστικά του (optimizer, μετρική σφάλματος) και εκτελούμε το training σε τρεις εποχές. Σε κάθε εποχή θέτουμε αρχικά το μοντέλο σε train mode και ύστερα για κάθε batch εκτελούμε την πρόβλεψη του μοντέλου για τα χαρακτηριστικά και υπολογίζουμε το σφάλμα που προκύπτει από τις αντίστοιχες τιμές των labels. Τέλος κάνουμε backpropagation για την ενημέρωση των βαρών του νευρωνικού.

Θέλουμε να εξασφαλίσουμε ότι και οι δύο κόμβοι ολοκλήρωσαν την εκπαίδευση και αυτό το επιτυγχάνουμε με χρήση barrier. Ακολουθεί το testing από τον master κόμβο (rank = 0), οπότε

```
######SIMPLE NN
class SimpleClassifier(nn.Module):
    def __init__(self, input_size):
        super(SimpleClassifier, self).__init__()
        self.fc = nn.Linear(input_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc(x)
        x = self.sigmoid(x)
        return x
```

Εικ. 1.Το νευρωνικό δίκτυο που υλοποιήσαμε

```
def train_epoch(dataloader, model, criterion, optimizer, device):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = criterion(pred, y.view(-1, 1))

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print('Loss: ',loss.item())
```

Εικ. 2. Η συνάρτηση που υλοποιεί το training, καλείται για κάθε εποχή

```
Training started with configuration:
 Training config
 train_loop_config/batch_size    128
 train_loop_config/epochs          3
 train_loop_config/lr           0.01
```

Εικ. 6. Οι παράμετροι του training loop.

ελέγχουμε την ακρίβεια του νευρωνικού μας με βάση τις προβλέψεις του για το test set.

Η διανομή των διεργασιών του συστήματος στους δύο κόμβους επιτυγχάνεται με την εξής μεθοδολογία: ορισμός distributed.init_process_group(), χρήση Distributed Sampler για να διανεμηθούν τα batches, wrap το μοντέλο με DistributedDataParallel για να γίνεται η ενημέρωση των βαρών και από τους δύο κόμβους, χωρίς συγκρούσεις και ασυνεπή αποτελέσματα.

```
dist.init_process_group("gloo", rank=rank, world_size=world_size)
```

```
train_sampler = DistributedSampler(celestial_dataset, num_replicas=world_size, rank=rank)
dataloader = DataLoader(celestial_dataset, batch_size=128, shuffle=False, sampler=train_sampler)
```

```
model = DistributedDataParallel(model)
```

Εικ. 3, 4, 5. Τα σημεία του κώδικα που επιτελούν την κατανομή

Ray: Τώρα ο decorator @ray.remote μας επιτρέπει να κάνουμε και την φόρτωση του Dataset με κατανεμημένο τρόπο . Ο βασικός κώδικας που αφορά το training και το testing παραμένει ο ίδιος, εκτός από την τροποποίηση: 'dataloader.sampler.set_epoch(epoch)' στην συνάρτηση για το training, που διασφαλίζει τον σωστό συγχρονισμό στην ενημέρωση των παραμέτρων του μοντέλου.
Σε αυτή την περίπτωση, η διανομή των διεργασιών του συστήματος στους διαθέσιμους κόμβους απαιτεί τις εξής τροποποιήσεις: αντί για χρήση distributed sampler, θα κάνουμε wrap τον dataloader με την συνάρτηση "prepare_data_loader()", ώστε κάθε worker να λάβει ένα διαφορετικό υποσύνολο των δεδομένων. Ομοίως και για το model.

```
dataloader= ray.train.torch.prepare_data_loader(dataloader)
model = ray.train.torch.prepare_model(model)
```

Εικ 7, 8. Τα σημεία του κώδικα που επιτελούν την κατανομή.

Και με τις δύο υλοποιήσεις πετυχαίνουμε **Accuracy=93.7%** στην ταξινόμηση του test set.

B. Personalised Page Rank σε κατευθυνόμενο γράφο

Στην συνέχεια επιλέξαμε να αξιολογήσουμε τις δυνατότητες των συστημάτων μέσω της εφαρμογής Personalised Page Rank σε κατευθυνόμενο γράφο. Προμηθευτήκαμε το dataset από το Stanford Network Analysis Platform (SNAP) [2]. Πρόκειται για έναν γράφο με 41652230 κόμβους και 1468364884 ακμές, που αφορά της “follows” σχέσεις μεταξύ των χρηστών του twitter (X) για το έτος 2010.

Αρχικά θα πούμε λίγα λόγια για τον αλγόριθμο. Ο αλγόριθμος PageRank είναι ένα παράδειγμα αλγορίθμου αξιολόγησης της σημαντικότητας κόμβων σε ένα δίκτυο. Λειτουργεί με βάση την ιδέα ότι η σημαντικότητα ενός κόμβου μπορεί να καθοριστεί από το πόσοι άλλοι κόμβοι συνδέονται μαζί του και πόσο σημαντικοί είναι αυτοί οι κόμβοι. Με άλλα λόγια, ένας κόμβος θεωρείται πιο σημαντικός και λαμβάνει υψηλότερο PageRank αν έχει πολλούς συνδέσμους από άλλες σημαντικούς κόμβους. Ο αλγόριθμος Personalised PageRank, που χρησιμοποιήσαμε εμείς πρόκειται για μια παραλλαγή, που παρέχει αποτελέσματα για κάθε κόμβο ξεχωριστά. Δηλαδή η εφαρμογή του με είσοδο τον node1 και όλον τον γράφο θα δώσει ως αποτέλεσμα τιμές που δείχνουν πόσο σημαντικός είναι ο κάθε κόμβος του γράφου για τον node1. Στην υλοποίησή μας θα υπολογίζονται παράλληλα τα Personalised PageRank scores για διαφορετικούς κόμβους του γράφου.

Pytorch: Δημιουργούμε το Dataset κατάλληλα, ώστε να έχει ως χαρακτηριστικά μια λίστα με τους κόμβους του γράφου (για τους οποίους θα εκτελέσουμε το PPR) και ένα tensor με τις ακμές του γράφου. Η __getitem__() θα επιστέφει κόμβους, καθώς αυτό θέλουμε να παραλληλοποιηθεί στα batches του dataloader. Για οικονομία χρόνου θα

```
def distributed_ppr(rank, world_size, node_dataset, dataloader):
    device="cpu"
    all_ppr_scores = []
    for batch in dataloader:
        batch=batch.to(device)
        batch_scores = personalized_page_rank(edge_index=node_dataset.edge_index,indices= batch)
        all_ppr_scores.extend(batch_scores)

    all_ppr_scores = torch.stack(all_ppr_scores)
    gathered_scores = [torch.zeros_like(all_ppr_scores) for _ in range(world_size)]
    dist.all_gather(gathered_scores, all_ppr_scores)
    # Combine gathered results
    if rank == 0:
        combined_results = torch.cat(gathered_scores, dim=0)
        return combined_results
    else:
        return None  # Non-root processes return None
```

Εικ. 9. Η συνάρτηση που υλοποιεί το PPR

εφαρμόσουμε τον αλγόριθμο στους 4 πρώτους κόμβους του γράφου και θα επιλέξουμε batch_size=2. Χρησιμοποιούμε την συνάρτηση personalised_page_rank() της torch_ppr βιβλιοθήκης. Η λογική είναι η εξής: υπολογίζει κάθε διεργασία το PPR για τους κόμβους του batch που της έχει αποδοθεί και ύστερα, μιας και τα αποτελέσματα για κάθε κόμβο είναι ανεξάρτητα από τους άλλους, με την all_gather() συνενώνουμε στον master (rank=0) τα επιμέρους αποτελέσματα. Αυτό μαζί με την χρήση του DistributedDataSampler καθιστούν το σύστημα κατανεμημένο.

Ray: Η λογική είναι η εξής: αφού αρχικοποιήσουμε το σύστημα και δημιουργήσουμε το dataset και τον dataloader, με τρόπο παρόμοιο με παραπάνω, θα καλέσουμε την τροποποιημένη συνάρτηση distributed_ppr() με παραμέτρους τις ακμές του γράφου (αποθηκευμένες στο χαρακτηριστικό edge_index του dataset) και ένα batch. Το batch αυτό θα περιέχει για κάθε διεργασία τους κόμβους για τους οποίους θα υπολογίσει τα personalised page rank scores. Η τροποποιημένη συνάρτηση γίνεται:

```
@ray.remote
def distributed_ppr(edge_index, nodes):
    device = "cpu"
    all_ppr_scores = []
    for node in nodes:
        node = torch.tensor([node]).to(device)
        batch_scores = personalized_page_rank(edge_index=edge_index, indices=node)
        all_ppr_scores.extend(batch_scores)
    return torch.stack(all_ppr_scores)
```

Εικ. 10. Η συνάρτηση που υλοποιεί το PPR

Στην συγκεκριμένη περίπτωση, η κατανομή στους πόρους επιτυγχάνεται ως εξής:

```
futures = [distributed_ppr.remote(node_dataset.edge_index, batch) for batch in dataloader]
#block until execution of all tasks is finished
results = ray.get(futures)
# Combine results
combined_results = torch.cat(results, dim=0)
```

Εικ. 11. Τα σημεία που επιτελούν την κατανομή στους πόρους

Ειδικότερα, η λίστα ‘futures’ δημιουργείται από non -blocking κλήσεις της distributed_ppr(), την οποία ορίζουμε ως κατανεμημένη. Η ray_get() μπλοκάρει την συνέχεια του προγράμματος έως ότου ολοκληρωθούν όλες οι κλήσεις της distributed_ppr() και συλλέγει τα αποτελέσματά τους.

Τα αποτελέσματα του αλγορίθμου που παίρνουμε σε κάθε περίπτωση είναι:

```
tensor([[9.1777e-02, 4.3671e-02, 0.0000e+00,  ..., 0.0000e+00, 0.0000e+00,
         1.4699e-08],
        [0.0000e+00, 0.0000e+00, 5.1287e-01,  ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00],
        [8.7338e-02, 9.1644e-02, 0.0000e+00,  ..., 0.0000e+00, 0.0000e+00,
         1.3973e-08],
        [0.0000e+00, 0.0000e+00, 4.8713e-01,  ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00]])
torch.Size([4, 41651670])
```

Εικ. 12. Αποτέλεσμα του PPR για τους 4 κόμβους

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)  # EMNIST images are 28x28
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)  # Output layer for 10 digits

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten the images
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)
```

Εικ. 13. Το νευρωνικό που χρησιμοποιήσαμε.

```
Train Epoch: 0 [0/240000 (0%)]  Loss: 2.311425
Train Epoch: 0 [64000/240000 (27%)]     Loss: 0.048702
Train Epoch: 1 [0/240000 (0%)]  Loss: 0.161504
Train Epoch: 1 [64000/240000 (27%)]     Loss: 0.014931
Train Epoch: 2 [0/240000 (0%)]  Loss: 0.073425
Train Epoch: 2 [64000/240000 (27%)]     Loss: 0.007925
Test set: Average loss: 0.0707, Accuracy: 39157/40000 (98%)
```

Εικ, 14. Αναλυτική παρακολούθηση της εξέλιξης της εκτέλεσης.

Πετυχαίνουμε ακρίβεια 98%

C. Classification σε image data

Το τελευταίο πείραμα αφορά την ταξινόμηση, όχι δυαδική αυτήν την φορά, εικόνων. Συγκεκριμένα θα χρησιμοποιήσουμε από το EMNIST dataset [3] το υποσύνολο που αφορά τα χειρόγραφα αριθμητικά ψηφία. Σκοπός μας είναι τα ταξινομήσουμε τις εικόνες σε 1 από τα 10 ψηφία.

Οι εικόνες του dataset αποτελούνται από 28x28 pixel σε grayscale. Παρέχονται σε IDX format (idx1 για labels και idx3 για images), το οποίο έχει τις εξής ιδιαιτερότητες: τα πρώτα 4 byte του αρχείου είναι το magic number (χρησιμοποιείται ως id), τα επόμενα 4 είναι το πλήθος των images/ labels και για το idx3 τα επόμενα 8 δείχνουν το πλήθος των γραμμών και των στηλών για κάθε εικόνα. Ακολουθούν τα πραγματικά δεδομένα. Προφανώς ο ορισμός του Dataset για την αρχική φόρτωση των δεδομένων διαφέρει από την περίπτωση του csv αρχείου. Το ενδιαφέρον, ωστόσο, είναι ότι άπαξ και φορτώσουμε τα δεδομένα σε tensors, ο χειρισμός τους στο training και testing παραμένει ακριβώς ο ίδιος. Μιας και το συγκεκριμένο dataset είναι ελαφρώς μικρότερο, θα χρησιμοποιήσουμε ένα πιο περίπλοκο νευρωνικό με τρία πλήρως συνδεδεμένα επίπεδα και ReLU συνάρτηση ενεργοποίησης.

IV. Αποτελέσματα – Συμπεράσματα

A. Χρόνος

Αρχικά θα προβούμε σε αξιολόγηση των λογισμικών Ray, Pytorch σχετικά με τις δυνατότητές τους σε ταχύτητα στην φόρτωση των δεδομένων και στην εκτέλεση του κώδικα.

Θα ξεκινήσουμε με το πρώτο πείραμα, δοκιμάζοντας διαφορετικά μεγέθη εισόδου: I1α → 100000 γραμμές, I1β → $4*10^6$ γραμμές.

	I1α	**I1β**
Pytorch	2’ 25”	1h 25’
Ray	2’ 50”	1h 35’

Πιν. 1. Μετρήσεις για BC για διάφορα μεγέθη εισόδου

Μια ακόμη μέτρηση που μπορούμε να κάνουμε αφορά το πλήθος των διεργασιών που θα τρέξουν παράλληλα και των πόρων του συστήματος που θέλουμε να χρησιμοποιήσουμε (π.χ. num_cpus). Το ray κατανέμει τα tasks σε διεργασίες με αυτόματο τρόπο, οπότε δεν υπάρχει έλεγχος από τον χρήστη. Στο Pytorch, από την άλλη, δεν υπάρχει αυτός ο περιορισμός και λαμβάνουμε τα εξής αποτελέσματα:

	2 processes	4 processes
I1α	2’ 25”	1’ 31”
I1β	1h 25’	1h 10”

Πιν. 2. Μετρήσεις για BC με Pytorch για διαφορετικό πλήθος workers

Όπως είναι αναμενόμενο, η αύξηση του παραλληλισμού βελτιώνει σημαντικά τον χρόνο εκτέλεσης.

Στην συνέχεια, θα εκτελέσουμε παρόμοιες μετρήσεις για το δεύτερο πείραμα. Θα χρησιμοποιήσουμε ως μεγέθη εισόδου τα: I2α → 100000 ακμές, I2β → 10^6 ακμές, I2γ → $7*10^6$ ακμές. Δεν δοκιμάζουμε να φορτώσουμε όλον τον γράφο, καθώς το μέγεθός του είναι απαγορευτικό για τις δυνατότητες των πόρων μας και ξεφεύγει από τις ανάγκες της εργασίας. Λαμβάνουμε:

	I2α	**I2β**	**I2γ**
Pytorch	6’ 25”	15’ 17”	32’ 20”
Ray	5’ 15”	11’ 37”	26’ 50”

Πιν. 3. Μετρήσεις για PPR για διάφορα μεγέθη εισόδου

	2 processes	**4 processes**
I2α	6’ 25”	4’ 10”
I2β	15’ 17”	9’ 40”
I2γ	32’ 20”	21’ 45”

Πιν. 4. Μετρήσεις για PPR με Pytorch για διαφορετικό πλήθος workers

Για το τρίτο πείραμα θα χρησιμοποιήσουμε: I3α → 240000 εικόνες και I3β → 100000 εικόνες στο train set:

	I3α	**I3β**
Pytorch	5’ 33”	3’ 40”
Ray	6’ 10”	4’ 38”

Πιν. 5. Μετρήσεις classification σε Images για διάφορα μεγέθη εισόδου

Παρατηρούμε πως στην περίπτωση του Clssification το Pytorch δίνει καλύτερους χρόνους εκτέλεσης. Αυτό συμβαίνει χάρη στο DDP πακέτο, που είναι βελτιστοποιημένο για τέτοιες εφαρμογές Deep Learning. Το Ray, από την άλλη, προσφέροντας ένα γενικότερο πλαίσιο κατανεμημένου υπολογισμού, δεν αποδίδει τόσο καλά στην συγκεκριμένη περίπτωση. Δίνει, ωστόσο, καλύτερους χρόνους για το PPR. Ακόμη, παρατηρούμε πως το Ray διέπεται από ένα όχι αμελητέο overhead στην αρχικοποίηση, κάτι που γίνεται περισσότερο αισθητό στα μικρά μεγέθη εισόδου.

Σε κάθε περίπτωση, η αύξηση του πλήθους των workers, που θα συνδράμουν για την εκτέλεση του κώδικα, βελτιώνει σε πολύ μεγάλο βαθμό τον χρόνο εκτέλεσης. Αυτό οφείλεται για το πρώτο πείραμα στην χρήση του DDP, που αναλαμβάνει την σωστή ενημέρωση των παραμέτρων του μοντέλου και για το δεύτερο πείραμα στην ανεξαρτησία των διαφορετικών επαναλήψεων. Η δυνατότητα ρύθμισης του πλήθους των workers, ανάλογα με τα όρια των πόρων του συστήματος, είναι ένα σημαντικό πλεονέκτημα του Pytorch έναντι του Ray.

B. Κλιμακωσιμότητα

Αφού έχουμε εγκαταστήσει τα προγράμματα και τις απαραίτητες βιβλιοθήκες, τρέχουμε τον κώδικα σε Pytorch ως εξής:

“

torchrun --nnodes=2 –node_rank=(0 ή 1) –master_addr=<private IP of master> --master_port=<random unused port> filename.py

”

Χρειάζεται να την τρέξουμε σε κάθε κόμβο, παραμετροποιώντας κάθε φορά το ‘node_rank’ (ο master θα περιμένει να συνδεθούν όσοι κόμβοι προσδιορίζονται από την παράμετρο nnodes). Αυτό σημαίνει πως το filename.py θα πρέπει να βρίσκεται σε κάθε κόμβο και κάθε ενδεχόμενη ενημέρωση του από έναν κόμβο θα πρέπει να μεταφέρεται χειροκίνητα στους άλλους.

Αντίθετα στο Ray, αφού συνδέσουμε όλους τους κόμβους στο ray cluster, όπως περιγράφηκε στο I.C, αρκεί να τρέξουμε στον master:

“ python3 filename.py ”

Το Ray θα αναλάβει την διανομή στους κόμβους. Επομένως, το Pytorch είναι απαγορευτικό για cluster με πολλούς κόμβους, ενώ το Ray προσφέρει κλιμακωσιμότητα.

C. Ευκολία χρήσης

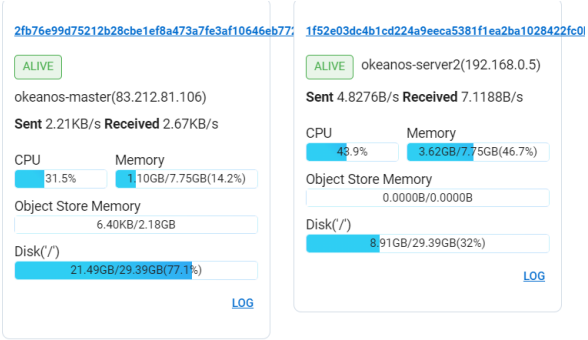
Άλλος ένας σημαντικός παράγοντας στην σύγκριση των δύο λογισμικών είναι η ευκολία χρήσης.

Στο (I) περιγράφηκε η διαδικασία εγκατάστασης και διαμόρφωσης του περιβάλλοντος για τις δύο περιπτώσεις. Από αυτά συμπεραίνουμε πως το Pytorch είναι ελαφρώς απλούστερο σε αυτόν τον τομέα.

Όσον αφορά στο κομμάτι του κώδικα, το Ray υπερτερεί στην ευκολία:. Χρειάζεται απλά να χρησιμοποιήσουμε τον διακοσμητή ‘ @ray.remote ’, για να δηλώσουμε τις συναρτήσεις που θέλουμε να εκτελέσουμε με κατανεμημένο τρόπο, να της καλέσουμε με ‘ func.remote() ’ και να συλλέξουμε τα αποτελέσματα με ‘ ray.get() ’. Το Ray αναλαμβάνει τον συγχρονισμό και την κατανομή στους πόρους. Ακολουθώντας αυτά τα βήματα μπορούμε να εκτελέσουμε οποιαδήποτε συνάρτηση με κατανεμημένο τρόπο. Αντίθετα το Pytorch έχει δυνατότητες, που περιορίζονται στο Deep Learning. Για άλλες εφαρμογές προσφέρει παράλληλη εκτέλεση σε batches ενός dataloader, όμως τότε αναλαμβάνει ο προγραμματιστής τον συγχρονισμό με χρήση barriers, όπου είναι δυνατόν.

Τέλος, αξίζει να αναφερθούμε και στις δυνατότητες για visualisation κάθε λογισμικού. Το Ray προσφέρει το Ray Dashboard, όπου μπορούμε να παρακολουθούμε την κατάσταση του cluster (dead or alive nodes), την χρήση των πόρων και την εκτέλεση των διεργασιών.

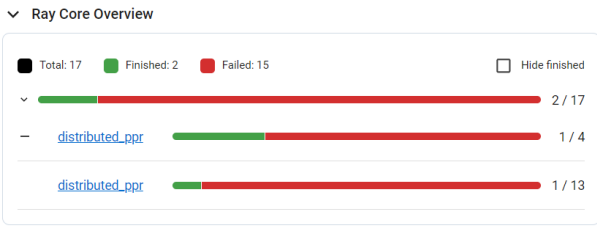
- π.χ.1 Στην παρακάτω εικόνα βλέπουμε πως πράγματι και οι δύο κόμβοι συμμετέχουν στους υπολογισμούς:



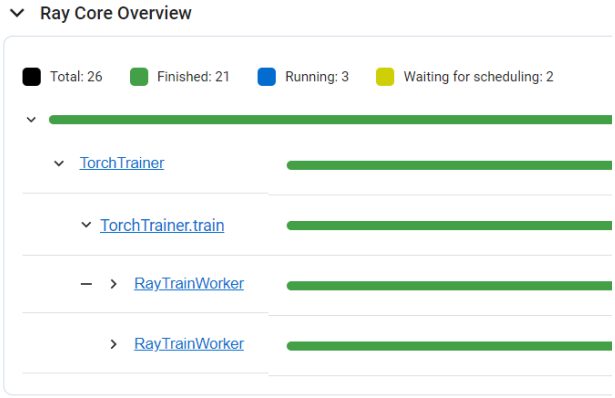
Εικ. 15. Κατάσταση των κόμβων από το Ray Dahboard

- π.χ.2 Στην εικόνα 14 βλέπουμε πως υπάρχει κάποιο πρόβλημα με την εκτέλεση του PPR, καθώς, προκειμένου να πετύχουν δύο διεργασίες, ξεκίνησαν άλλες 15, οι οποίες απέτυχαν. Αξίζει να σημειώσουμε πως, παρά την αρχική αποτυχία των διεργασιών, το σύστημα συνέχισε να προσπαθεί, ξεκινώντας νέες.

- π.χ.3 Στην εικόνα 15 μπορούμε να δούμε δύο instances του RayTrain, που τρέχουν παράλληλα.



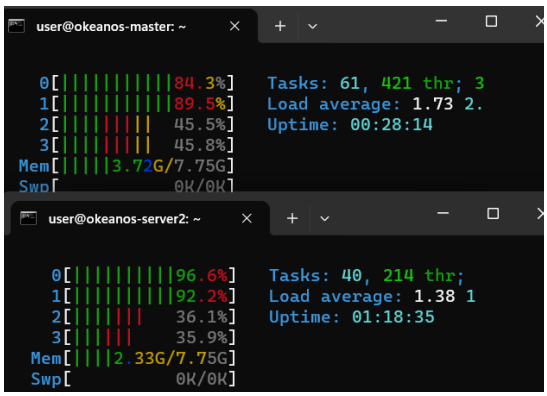
Εικ. 16 Παρακολούθηση ενός Job για PPR από το Dashboard



Εικ. 17. Παρακολούθηση ενός Job για BC από το Dashboard

Συγχρόνως το Ray Dashboard προσφέρει την δυνατότητα σύνδεσης με τα Prometheus και Grafana, τα οποία προσφέρουν οπτικοποίηση ακόμη περισσότερων χαρακτηριστικών του συστήματος και της εκτέλεσης του κώδικα.

Αντίθετα το Pytorch δεν προσφέρει καμία τέτοια δυνατότητα. Ο μόνος τρόπος να παρακολουθούμε την εκτέλεση είναι τρέχοντας ‘htop’ στους κόμβους.



Εικ. 18. Οπτικοποίηση μέσω htop.

Συμπέρασμα:

Τα Pytorch, Ray είναι δύο λογισμικά με αρκετές δυνατότητες, που σχεδιάστηκαν για διαφορετικούς σκοπούς. Επιπλέον, το PyTorch έχει εξελιχθεί ως ισχυρό εργαλείο στον χώρο της Επιστήμης των Δεδομένων και της Τεχνητής Νοημοσύνης, παρέχοντας ευελιξία στην κατασκευή, τον πειραματισμό και την εκπαίδευση μοντέλων βαθιάς μάθησης. Έχει γίνει ευρέως διαδεδομένο στην κοινότητα της Τεχνητής Νοημοσύνης, και πολλά εργαλεία και βιβλιοθήκες υποστηρίζουν το PyTorch. Το Pytorch προσφέρει βιβλιοθήκες για Deep Learning και βελτιστοποιεί την κατανομή της εκπαίδευσης για τέτοιες εφαρμογές. Είναι πιο low-level, για αυτό κι επιτρέπει στον χρήστη την ρύθμιση κάποιων παραμέτρων της κατανομής. Από την άλλη πλευρά, το Ray έχει σχεδιαστεί για να διευκολύνει την ανάπτυξη και την κλιμάκωση κατανεμημένων εφαρμογών. Με τη χρήση του Ray, οι προγραμματιστές μπορούν να δημιουργήσουν κατανεμημένα συστήματα με μικρές τροποποιήσεις στον αρχικό κώδικα τους, επιτρέποντας τους να επωφεληθούν από τα πλεονεκτήματα της κλιμάκωσης σε μεγάλα cluster. Επιπλέον, το Ray παρέχει εργαλεία για τον προγραμματισμό της ροής εργασιών (workflow) και την οπτικοποίηση των εκτελεστικών διεργασιών.

Συνολικά, τόσο το PyTorch όσο και το Ray συμβάλλουν στην εξέλιξη της επιστήμης των υπολογιστών, καλύπτοντας διάφορες ανάγκες στους τομείς της μηχανικής μάθησης, της κατανεμημένης υπολογιστικής, και της ανάλυσης δεδομένων. Πρόκειται για σχετικά νέα λογισμικά (2017, 2016), πολλά υποσχόμενα για τον τομέα της Μηχανικής Μάθησης.

V. Κώδικας

Τον κώδικα που υλοποιήσαμε θα τον βρείτε στο παρακάτω Github link:

https://github.com/georgia-bous/InformationSystems-Pytorch_vs_Ray

VI. Αναφορές

[1] https://www.kaggle.com/datasets/hari31416/celestialclassify

[2] https://snap.stanford.edu/data/twitter-2010.html

[3]Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373