МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ІВАХІВ Віталій Володимирович

Методи та засоби інтеграції розподілених систем обробки великих даних з моделями глибокого навчання / Methods and Tools for Integration of Distributed Big Data Processing Systems with Deep Learning Models

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп’ютерні науки

Кваліфікаційна робота

Виконав студент групи
КНм-21
В.В. Івахів
_____

Науковий керівник:
д.т.н., доцент М.П. Комар
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
Завідувач кафедри
_____ М.П. Комар

ТЕРНОПІЛЬ - 2021

ЗМІСТ

ВСТУП

Актуальність теми. З появою цифрових технологій та розумних пристроїв величезна кількість даних генерується з величезною швидкістю. За даними Агентства національної безпеки, Інтернет обробляє 1826 петабайт даних на добу [1]. З огляду на нові технології та дані, які генеруються всіма пристроями IoT (Інтернет речей), передбачається, що протягом наступних кількох років буде створено більше обсягів даних, ніж зараз обробляється. Насправді, близько 90% поточних даних було створено за останні декілька років, і вони будуть продовжувати зростати в найближчому майбутньому [1].

Експоненційний ріст цифрових даних, які генеруються з численних і різноманітних джерел, робить неможливим зберігання, обробку та аналіз за допомогою традиційних методів. Ці обмеження призвели до еволюції технологій навколо великих даних. Великі дані, які визначаються швидким зростанням обсягу, різноманітності та швидкості даних, як правило, мають справу з неструктурованими даними, які потребують великого пакетного аналізу або аналізу в реальному часі.

Крім величезного обсягу великих даних і постійним їх зростанням [42], складна структура цих нових даних і складність управління і захисту таких даних додали додаткових проблем. З тих пір, як виникла ідея великих даних, вони стали одним з найпопулярніших напрямків як в технічній, так і в інженерної областях [43]).

Глибоке навчання (Deep Learning, DL), підмножина машинного навчання, – це технологія, яка використовується для аналізу й обробки величезної кількості даних, щоб допомогти знайти абстрактні й корисні моделі [14]. Глибоке навчання навчається на різних рівнях представлень і абстракцій будь-яких даних, таких як текст, зображення, звук, відео, часові ряди тощо і є ефективним засобом обробки таких даних.

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення ефективності обробки великих даних на основі інтеграції

розподілених систем обробки великих даних з моделями глибокого навчання.

Для досягнення поставленої мети у роботі сформульовано наступні задачі:

– провести аналіз предметної області;

– провести аналіз розподілених систем обробки великих даних;

– зробити постановку задач дослідження;

– дослідити нейромережеві моделі глибокого навчання для обробки великих даних;

– дослідити методи реалізації моделей глибокого навчання для обробки великих даних;

– розробити метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання;

– дослідити інструментальні засоби попередньої обробки великих даних;

– провести реалізацію програмного забезпечення для інтеграції великих наборів даних та моделей глибокого навчання;

– провести оцінку ефективності запропонованого методу інтеграції великих наборів даних та моделей глибокого навчання.

Об’єкт дослідження – процеси обробки великих даних в автоматизованих системах обробки інформації.

Предмет дослідження – методи та засоби інтеграції систем обробки великих даних та моделей глибокого навчання.

Методи дослідження. Для вирішення поставлених задач в роботі використовувалися наступні методи великих даних, розподіленої обробки даних, попередньої обробки даних, а також методи глибоких нейронних мереж.

Наукова новизна одержаних результатів. Вдосконалено метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання, що дозволить трансформувати дані, що надходять з різних потоків

великих даних, у формат, необхідний для навчання глибоких нейронних мереж, а також дозволить виконувати різні маніпуляції з потоками даних.

Практичне значення отриманих результатів. Розроблено архітектуру програмного забезпечення інтеграції систем обробки великих даних та моделей глибокого навчання. Програмне забезпечення надає користувачеві можливість вибору даних (пакетних даних / даних у реальному часі) з різних архітектур великих даних, попередньої обробки даних для перетворення даних у необхідний формат, а потім навчання моделей глибокого навчання з використанням попередньо оброблених даних.

Публікації та апробація. Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах:

– X міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій», Тернопіль, Тернопільський національний технічний університет імені І. Пулюя, 2021 р.

– міжнародної наукової Інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», 2021 р.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ТЕХНОЛОГІЙ ОРГАНІЗАЦІЇ РОЗПОДІЛЕНОЇ ОБРОБКИ ВЕЛИКИХ ДАНИХ

1.1 Аналіз предметної області обробки великих даних

Як правило, великі дані (Big Data) – це набори даних, які є дуже великими і складними для обробки та аналізу за допомогою традиційних інструментів. Такі набори даних відрізняються великою різноманітністю і швидкістю, що потребує розробки можливих рішень для отримання цінності і знань з таких широкомасштабних, і таких, що швидко змінюються даних [11, 14, 33, 51, 53].

Експоненційний ріст великих даних, які генеруються з різноманітних джерел, робить неможливим зберігання, обробку та аналіз за допомогою традиційних засобів. Ці обмеження призвели до еволюції технологій навколо великих даних. Великі дані, які визначаються швидким зростанням обсягу, різноманітності та швидкості даних, як правило, мають справу з неструктурованими даними, які потребують великого пакетного аналізу або аналізу в реальному часі [42].

Крім величезного обсягу великих даних і постійним їх зростанням, складна структура цих нових даних і складність управління і захисту таких даних додали додаткових проблем. З тих пір, як виникла ідея великих даних, вони стали одним з найпопулярніших напрямків як в технічній, так і в інженерної областях [43]).

Це створило величезну зміну парадигми в області виявлення та аналітичної обробки даних. Доведено, що аналіз великих даних корисний для різних секторів, таких як промисловість, охорони здоров'я, виробництво, підприємства, освітні послуги та швидко розширюється у всіх галузях науки і техніки, включаючи фізичні, біологічні та біомедичні науки [43].

Для отримання значних результатів з таких обсягів даних потрібна величезні витрати з точки зору зберігання та обчислювальних ресурсів, а також потрібні методи паралельної обробки великих наборів даних.

Машинне навчання (ML) – це галузь штучного інтелекту, яка виявляє закономірності шляхом аналізу великих наборів даних. Розуміння, отримане з використанням методів машинного навчання, є більш глибоким і дає результати у прискореному темпі та в масштабі, з яким люди не можуть зрівнятися [12]. Глибоке навчання (DL), підмножина машинного навчання, – це метод, що використовується для аналізу та обробки величезної кількості даних, щоб допомогти знайти з них абстрактні та корисні закономірності [14]. Глибоке навчання навчається на кількох рівнях представлень та абстракцій будь-яких даних, таких як текст, зображення, звук, відео, часові ряди тощо.

Якщо застосувати глибоке навчання до великих даних, то зможна знайти невідомі й корисні закономірності, які раніше було неможливо знайти [14].

З рисунку 1.1 видно, що традиційні підходи машинного навчання показують кращу продуктивність для меншої кількості вхідних даних. Оскільки обсяг даних перевищує певну кількість, продуктивність традиційних підходів машинного навчання стає стабільною, тобто досягає плато. Однак продуктивність підходів глибокого навчання збільшується по відношенню до збільшення обсягу даних [27].

Використання глибокого навчання для отримання важливої інформації з великих даних часто стає досить складним через наступне:

– обсяг: час виконання і складність моделі глибокого навчання збільшується із збільшенням кількості прикладів (входів), великої різноманітності типів класів (виходів), високої розмірності (атрибутів) та наявності шумних і неповних даних;

– різноманітність: дані надходять у всіх типах форматів із різних джерел і з різними розподілами;

– швидкість: дані генеруються з дуже високою швидкістю, тому потрібно обробляти пакетні потоки даних і потоки даних в режимі реального часу.

Типовим прикладом глибокого навчання з використанням великих даних є аналіз настроїв Twitter. Twitter публікує мільйони твітів на годину, що містять різну інформацію, поширюючи погляди на багато різних тем, як-от політика, бізнес, економіка тощо. Користувачі Twitter регулярно публікують свої думки щодо певної новини, нещодавно придбаного продукту чи послуги та, зрештою, про все, що відбувається навколо них. Тому сьогодні багато компаній проводять аналіз настроїв щодо твітів користувачів, щоб оцінити цінність своєї продукції.

Щоб провести такий наліз потрібно зібрати всі твіти про продукт, щоб провести аналіз настроїв. Потім попередньо обробляємо зібрані твіти, щоб видалити всі невідповідні дані. Після завершення попередньої обробки навчаємо моделі глибокого навчання, щоб отримати інформацію про продукт/політичні погляди/бізнес.

На рисунку 1.2 наведено огляд кроків використання даних Twitter із глибоким навчанням.

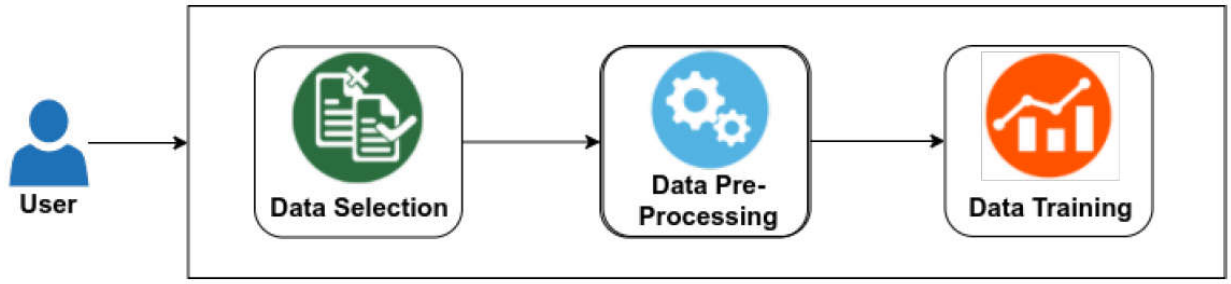

Рисунок 1.2 – Приклад аналізу настроїв у Twitter

1.2 Розподілена платформа Apache Hadoop

Apache Hadoop – це вільна програмна платформа і каркас для організації розподіленої обробки великих обсягів даних (що вимірюються у петабайтах) з використанням парадигми MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера [3, 20].

Hadoop складає собою платформу, що складається з декількох десятків модулів, які працюють разом з метою створення єдиного програмного каркасу.

Основними її модулями є [3, 20]:

Hadoop Common – містить бібліотеки та утиліти, необхідні для роботи інших модулів Hadoop;

Hadoop YARN – платформа для управління обчислювальними ресурсами та планування робіт;

Hadoop Distributed File System (HDFS) – розподілена файлова система, яка забезпечує високу агреговану пропускну здатність кластера;

Hadoop MapReduce – імплементація програмної моделі MapReduce для моделювання високомасштабної обробки даних.

HDFS.

Hadoop Distributed File System (HDFS) – це високомасштабована та розподілена файлова система, призначена для використання з Hadoop, та імплементована на Java. Кластер Hadoop базується на HDFS. У кластері є один простір імен та кілька вузлів даних. Файли, частини файлів, архіви зазвичай великі за розміром в HDFS. Розподіленість полягає у тому, що файли зберігаються на кількох вузлах. Стійкість до збоїв досягається завдяки реплікації між вузлами. Під час збоїв вузли можуть спілкуватися один з одним, і завдяки цьому можливе балансування даних у мережі.

Головною перевагою HDFS є те, що вона розроблена для запуску з урахуванням дешевого апаратного забезпечення. Це забезпечує високу

пропускну здатність та оброблення великих даних. З іншого боку, рваі підтримує доступ до даних із великими затримками. Це пояснюється тим, що хдаі розроблялася для пакетної обробки даних, а не для інтерактивної роботи.

Кластер може складатися із тисяч машини, що дозволяє зберігання та обробку великих масивів даних. Файли дозволяють тільки додавання у кінець, що означає, що оновлення даних неможливе. Це спрощує модель HDFS та уможливлює високу пропускну здатність. Останнє досягається обчисленням у тому ж місці, де знаходяться дані. За моделлю архітектури обчисленням відбувається на тому ж вузлі, де дані власне знаходяться, проміжні результати зберігаються на вузлах, де дані витягуються або повертають остаточне значення вузлу хазяїну, який запустив потік обчислень. Ця абстракція уникає витягування великих масивів даних через мережевий зв’язок та спрощує модель запуску додаткових обчислювальних ресурсів.

HDFS кластер складається з одного вузла імен, який вважається хазяїном, та вузлів даних, які вважаються рабами. Хазяїн керує простором імен кластера та доступом до вузлів даних. На практиці хазяїн відповідає за управління операціями над файлами, тобто відкриття, закриття, перейменування, додавання у кінець, управління блоками даних. Якщо файл зберігається у кластері HDFS, тоді він розподілений між вузлами кластера за блоками. Повна стурктура HDFS кластеру зображена на рисунку 1.2.

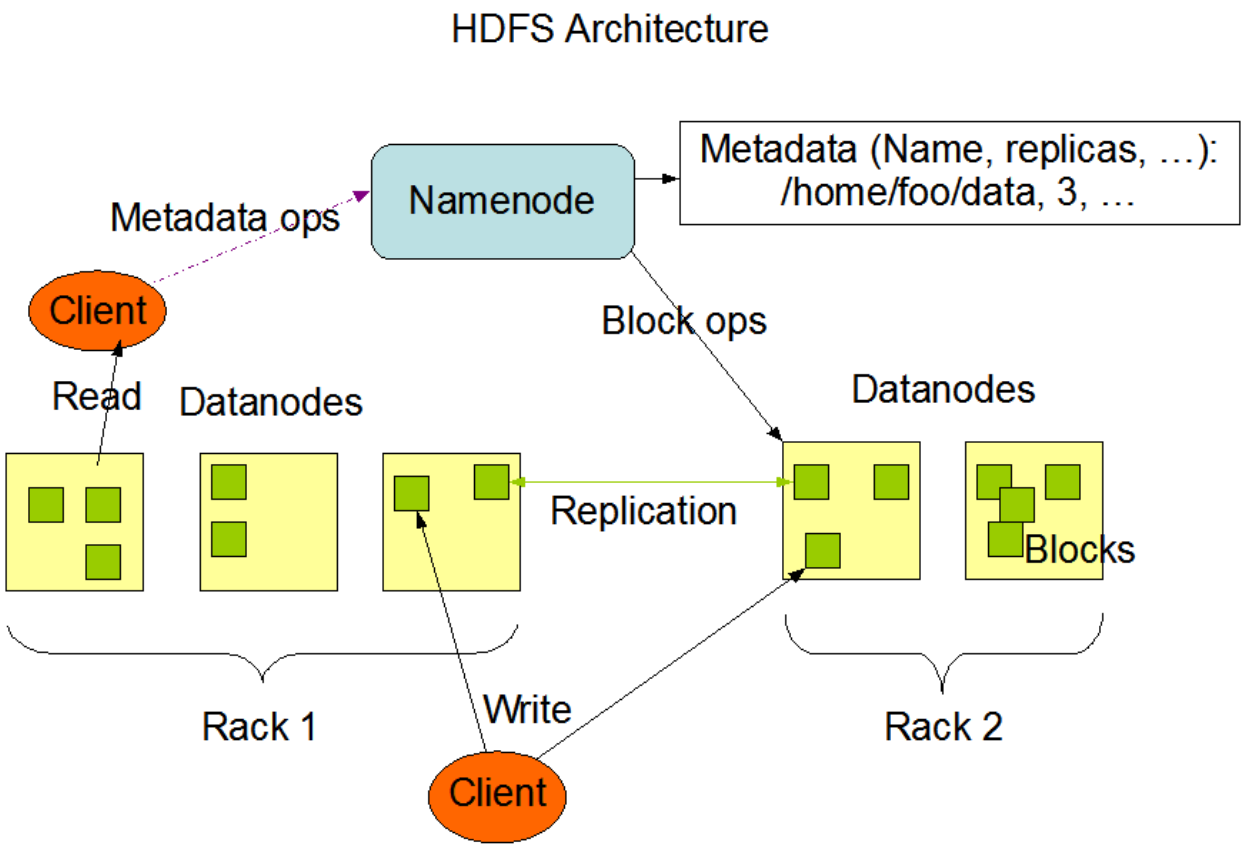

Рисунок 1.2 – Архітектура HDFS

Блоки в HDFS за замовчуванням мають розмір у 64 мегабайти, це значення змінне. Якщо файли, що зберігаються, менші за 64 мегабайти, має сенс змінити розмір блоку або навіть з'єднати менші файли у більші. Менше число великих файлів означає менше витрат на читання/запис, пам'яті на метадані тощо. Усі блоки за замовчуванням реплікуються, це також можна налаштувати у залежності від того, скільки реплікацій конкретного блоку зберігається. Інформація про реплікації важлива для вузла-хазяїна, який отримує періодичне оновлення статусу (серцебиття) від рабів. Якщо повідомлення про оновлення відсутнє протягом деякого часу, тоді вузол вважається мертвим без жодних операцій у майбутньому. Хазяїн відслідковує усі блоки, на які вплинула втрата вузла, і реплікація почнеться на іншому вузлі. Це стає можливим завдяки тому, що раби періодично відправляють звіти про блоки хазяїну, і ці метадані стають базою для рішень хазяїна про реплікацію.

Hadoop YARN.

Hadoop YARN – це система управління контейнерами та планувальник задач, створений здебільшого для використання з кластерами Hadoop.

Головною ідеєю YARN є розділення функціональності урпавлеіння ресурсами та планування/моніторингу робіт у різні демони. Ідея полягає в тому, щоб мати глобальний менеджер ресурсів (MP) та майстер застосунків для кожного застосунку. Застосунок – це або одинична робота, або граф робіт (workflow).

ResourceManager та NodeManage формують єдиний обчислювальний фреймворк. ResourceManager – це остаточна інстанція, яка розподіляє ресурси між застосунками в системі. ResourceManager – це фреймворк-агент, який створюєтся для кожної окремої машини і який відповідальний за контейнери, моніторинг використання ними ресурсів (процесор, пам’ять, диск, мережа) та звітування ресурсному менеджеру / планувальнику.

Ресурсний менеджер має дві головні компоненти: планувальник (Scheduler) та менеджер застосунків (ApplicationsManager). Архітектура YARN зображена рисунку 1.3.

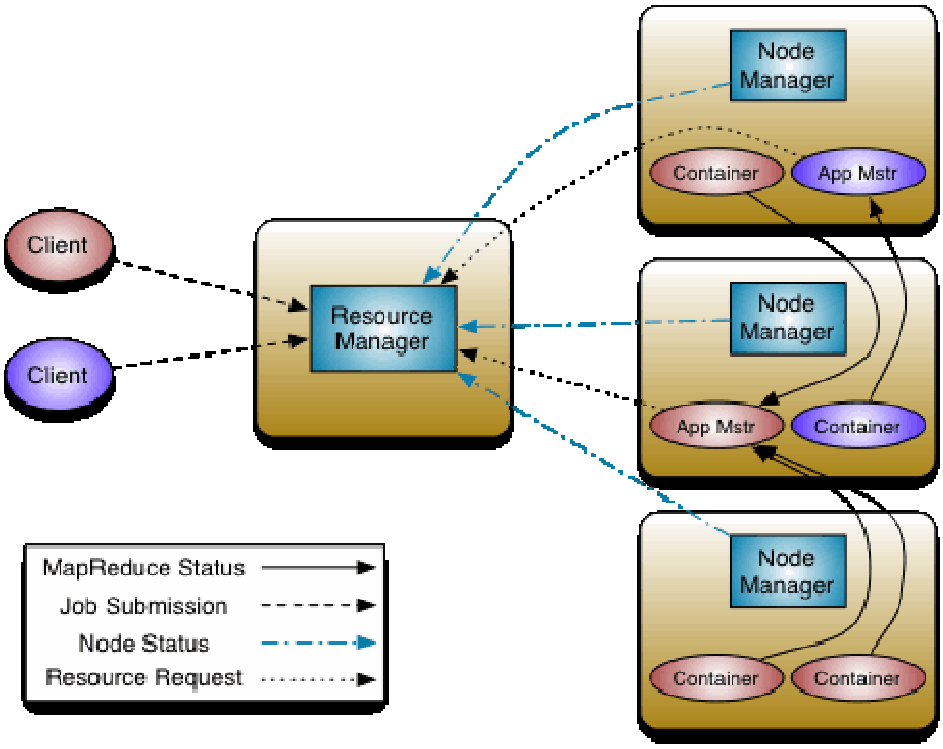

Рисунок 1.3 – Архітектура YARN

Планувальник відповідальний за розподіл ресурсів для різних застосунків, який враховує різноманітні обмеження на розмір ресурсів, черги і т.д. Він не виконує задач моніторингу або відслідковування статусу застосунків. Також, він не гарантує перезапуск робіт, які аварійно зупинилися (через збій у самому застосунку чи в апаратному забезпеченні). Планувальник має архітектуру, яка дозволяє зручне використання плагінів, які відповідальні за поділ ресурсів кластера між різними чергами, застосунками, групами користувачів, тощо. Прикладами таких плагінів є CapacityScheduler та FairScheduler.

ApplicationsManager відповідальний за запуск робіт, вибір контейнера для виконання конкретної задачі. Також він надає механізм відновлення та перезапуску задачі в разі збою у програмному чи апаратному забезпеченні, відслідковування статусу та прогресу задачі.

1.3 Розподілена платформа Apache Spark

Загальний огляд.

Apache Spark – це розподілений фреймворк кластерних обчислень для загальних цілей з відкритим кодом. Spark надає інтерфейс для програмування цілих кластерів з неявним паралелізмом даних та стійкістю до відмов [5].

Spark був розроблений у 2012 році як відповідь до обмежень парадигми кластерних обчислень MapReduce, яка строго задає особливу лінійну структуру потоку даних для розподілених програм: програми MapReduce читають вхідні дані з диску, застосовують (map) функцію на даних, зводять (reduce) результати роботи функції, та зберігають зведені результати на диску.

Spark полегшує розробку як ітеративних алгоримтів, які відвідують дані багато разів у циклі, так і інтерактивний/дослідницький аналіз даних, тобто повторюване звернення до даних у стилі баз даних. Час відповіді таких

застосунків може бути зменшений на кілька порядків у порівнянні із парадигмою MapReduce (зокрема, в стеку Apache Hadoop). Серед класу ітеративних алгоритмів є алгоритми тренування для систем машинного навчання, які сформували поштовх для розробки Apache Spark.

Apache Spark потребує кластерного менеджера та розподілену систему зберігання даних. Для кластерного менеджменту Spark підтримує самостійний режим (нативний кластер Spark), Hadoop YARN (який використовується в даній роботі) або Apache Mesos. Для розподіленого зберігання даних Spark може використовувати інтерфейси широкого різноманіття рішень, таких як HDFS, MapR File System, Cassandra, Openstack Swift, Amazon S3. Spark також підтримує псевдорозподілений локальний режим, який зазвичай використовується тільки для розробки або в цілях тестування, де розподілене збереження даних не вимагається і може бути використана локальна файлова система; у такому сценарії Spark запускається на одній машині з одним виконавцем на ядро. Архітектура Spark схематично зображена на рисунку 1.4.

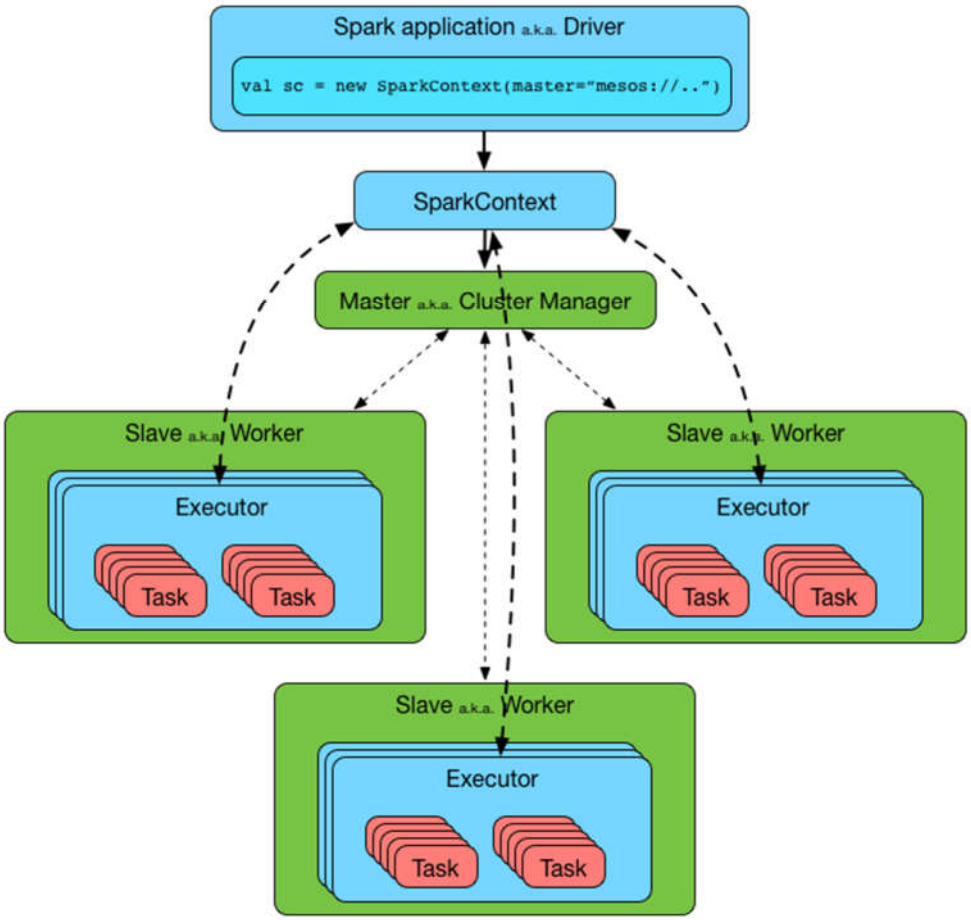

Рисунок 1.4 – Архітектура Spark

Spark Core.

Основою проекту Spark є підпроект Spark Core [5]. Він пропонує планування розподілених завдань та базову I/O функціональність, яка доступна через єдиний інтерфейс програмування застосунків (API), заснований на абстракції відмовостійкого розподіленого набору даних (Resilient Distributed Dataset, RDD). Цей інтерфейс повторює функціональну модель програмування: програма-драйвер викликає паралельній операції, такі як map, filter, reduce на RDD, передаючи функцію у Spark, далі планує розпаралелене виконання функції у кластері. Ці операції та додаткові, такі як join, беруть RDD на вхід та повертають новий RDD на виході. RDD незмінні та операції над ними ліниві (обчислення виконуються тільки коли результуючі дані запрошуються на вихід). Відмовостійкість досягається завдяки тому, що відслідковується «родовід» кожного RDD (послідовність

операцій, які створили даний RDD з початкового), так що в разі втрати даних цей набір можна відновити з родоводу. RDD можуть містити в собі будь-які об’єкти Scala, Java, Python. Взаємодія між RDD описана на рисунку 1.5.

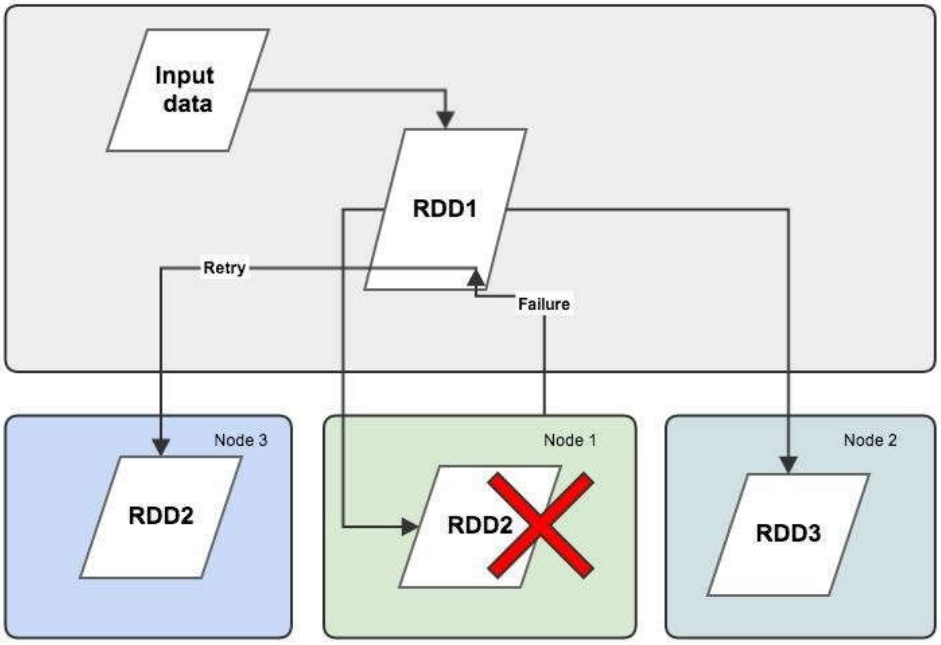

Рисунок 1.5 – Архітектура RDD

Окрім функціонального стилю програмування, орієнтованого на RDD, Spark пропонує дві форми спільних змінних:

– змінні трансляції містять у собі посилання на дані, доступні лише для читання, які необхідні для доступу на всіх вузлах. Якщо один набір даних використовується на всіх вузлах, це дозволяє підвищити ефективність завдяки тому, що дані копіюються на кожен вузол і не здійснюється зайвих запитів. Змінні трансляції не можуть змінюватися під час виконання програми.

– акумулятори використовується для агрегування загальної кількості даних та сум. Вузол хазяїн генерує порожні значення, а працівник заповнює їх та відправляє назад хазяїну, який є єдиною машиною, який може читати створені та заповнені дані.

Кластер Спарк створений для задач, які потребують низької затримки. Він може впоратися із задачею за мілісекунди у кластері з тисячма ядрами. Hadoop, на противагу, потребує 5-10 секунд для запуску будь-якої задачі.

Великою перевагою Spark над Hadoop є відсутніть реплікації в мережі, тому що навіть мережа на 10 гігабіт набагато швидше, ніж сучасна RAM. Spark тримає лише одну копію RDD у пам’яті. Якщо вузол повільний, копія бекапу запускається на іншому вузлі. Це досягається завдяки графу родоводу. Відновлення є швидким і всі втрачені частини даних одночасно перераховуються в залежності від кількості наявних вузлів.

Spark MLib – це розподілений фреймворк машинного навчання, який працює на основі Spark Core, і який, великою мірою в силу розподіленої архітектури, заснованої на пам’яті, Spark, майже в 9 разів швидше, ніж заснована на диску імплементації алгоритмів машинного навчання бібліотеки Apache Mahout.

Spark добре поєднується з фреймворком Hadoop – як файловою системою HDFS, так і планувальником задач YARN. Взаємодія із планувальником YARN зображена на рисунку 1.6.

Spark у поєданні з YARN є особливо ефективним через використання локальності даних: обробка та аналіз запускаються на тих вузлах, де розташовані дані, таким чином зменшуються затримки, пов’язані з копіюванням даних через мережу.

Система Hadoop MapReduce є набагато повільнішою за Spark, однак обидві можуть використовувати компонент збереження даних HDFS.

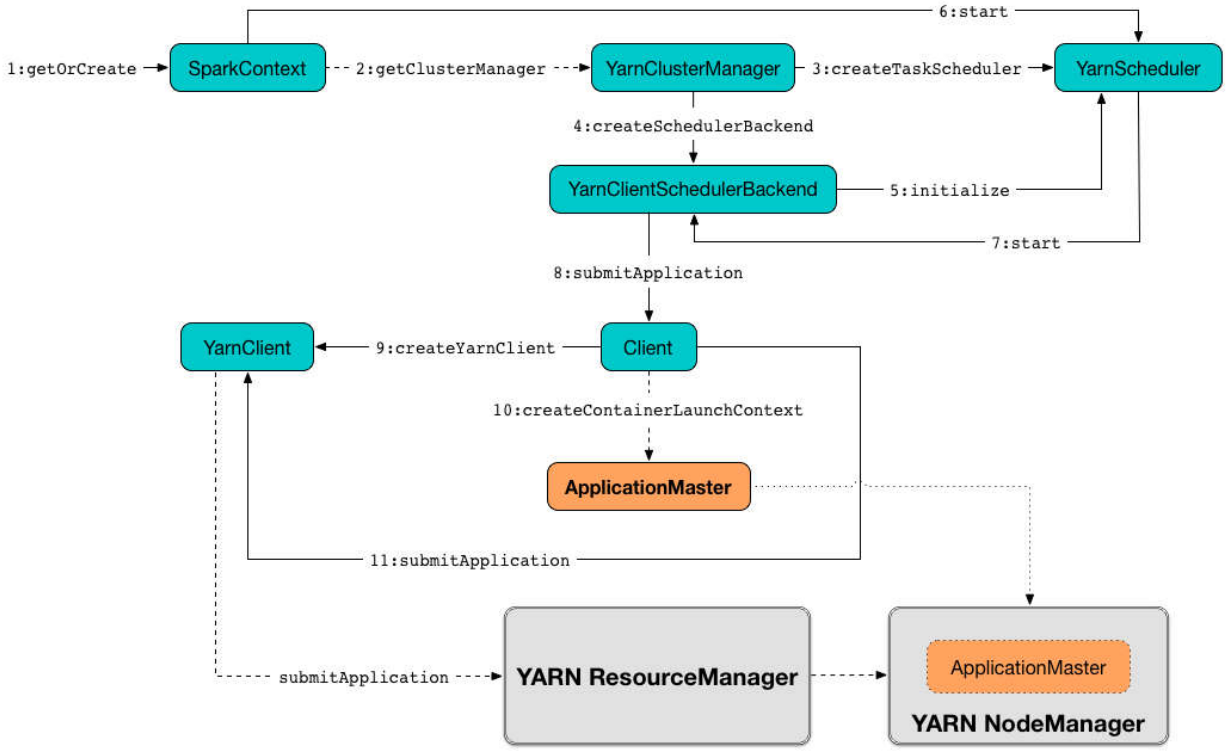

Рисунок 1.6 – Взаємодія Apache Spark та Apache YARN

Бібліотека GraphX.

Фундаментальним класом Spark є Resilient Distributed Dataset. Однак для роботи з графами було створено новий компонент Spark – бібліотеку GraphX. GraphX на високому рівні є розширенням RDD, який представляє собою нову абстракцію графу: орієнтований мультиграф із атрибутами для кожної вершини та ребра. Для підтримки обчислень на графах, GraphX використовує набір фундаментальних операторів (subgraph, joinVertices, aggregateMessages). Також GraphX включає ряд алгоритмів на графах та конструкторів для спрощення роботи з графовою аналітикою.

Основними класами GraphX, які використовуються в даній роботі, є класи Graph та Edge (ребро). Так само як і RDD, графи в GraphX незмінні, розподілені та відмовостійкі. Зміни до значення або структури графу виконуються шляхом створення нового графу з відповідними змінами. Так само, як із RDD, кожний розділ графу може бути відтворений на іншій машині в разі відмови.

Опишемо головні атрибути та методи цих класів (таблиця 1.1, таблиця 1.2).

Таблиця 1.1 – Клас spark.Edge

Атрибут/Метод	Опис
New Edge (srcId: VertexId = 0, dstId: VertexId = 0, attr: ED = null.asInstanceOf[ED])	Конструктор ребра, на вхід приймає параметри id вершин входу та виходу та атрибут ED, пов’язаний із ребром
var attr: ED	Атрибут, асоційований із ребром
var srcId: VertexId	Ідентифікатор вихідної вершини
var dstId: VertexId	Ідентифікатор вхідної вершини
def otherVertexId(vid: VertexId): VertexId	Маючи ідентифікатор однієї вершини, повертає іншу
def relativeDirection(vid: VertexId): EdgeDirection	Повертає відносний напрямок ребра до відповідної вершини

Таблиця 1.2 – Клас spark.Graph

Атрибут/Метод	Опис
New Graph (RDD <scala.Tuple2 <Object, Object> > rawEdges, VD defaultValue, scala.Option <PartitionStrategy> uniqueEdges, StorageLevel, edgeStorageLevel, StorageLevel vertexStorageLevel)	Конструктор графу, на вхід приймає: - rawEdges – колекція ребер у форматі (вихідне, вхідне); - defaultValue – атрибути вершин, з якими граф створюється за замовчуванням; - uniqueEdges – якщо кілька однакових ребер знайдено, вони об’єднується та атрибут ребра виставляється до суми. В іншому разі однакові ребра вважаються різними;

	- edgeStorageLevel/vertexStorageLevel – бажаний рівень сховища пам’яті, на якому кешуються ребра/вузли в разі необхідності
var attr: ED	Атрибут, асоційований із ребром
var srcId: VertexId	Ідентифікатор вихідної вершини
def vertices()	Зберігає стан графу в кеші для подальшої швидкої обробки
def vertices()	Повертає вершини графу
def edges()	Повертає ребра графу

1.4 Постановка задач дослідження

Аналіз систем розподіленої обробки великих даних показав їх обмеженість щодо ефективного спільного використання з нейромережевими моделями для аналізу та обробки великих обсягів даних. Відсутність програмного середовища для інтеграції систем обробки великих даних та моделей глибокого навчання спонукає пошук таких рішень.

Тому запропоновано використати підхід інтеграції систем обробки великих даних та моделей глибокого навчання, застосовуючи який користувачі зможуть трансформувати дані, що надходять з різних потоків великих даних, у формат, необхідний для навчання моделей глибокого навчання. Запропонований підхід забезпечить структуру, яка інтегрує різні потоки великих даних та моделі глибокого навчання, а також дозволить користувачам виконувати різні маніпуляції з потоками даних. Використовуючи таке рішення, користувачі зможуть швидко побудувати та виконати різні експерименти з підмножиною набору даних, щоб отримати з нього значущі результати.

Для реалізації такого підходу необхідно розробити програмне забезпечення, яке повинно надавати користувачеві можливість вибору даних (пакетних даних / даних у реальному часі) з різних архітектур великих даних, попередньої обробки даних для перетворення даних у необхідний формат, а потім навчання моделей глибокого навчання з використанням попередньо оброблених даних. Програмна система також повинна надавати можливість запису всіх виконаних кроків, а згодом дозволяти користувачеві запускати їх на всьому наборі даних.

Метою кваліфікаційної роботи є підвищення ефективності обробки великих даних на основі інтеграції розподілених систем обробки великих даних з моделями глибокого навчання.

Для досягнення поставленої мети у роботі сформульовано наступні задачі:

– провести аналіз предметної області;

– провести аналіз розподілених систем обробки великих даних;

– зробити постановку задач дослідження;

– дослідити нейромережеві моделі глибокого навчання для обробки великих даних;

– дослідити методи реалізації моделей глибокого навчання для обробки великих даних;

– розробити метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання;

– дослідити інструментальні засоби попередньої обробки великих даних;

– провести реалізацію програмного забезпечення для інтеграції великих наборів даних та моделей глибокого навчання;

– провести оцінку ефективності запропонованого методу інтеграції великих наборів даних та моделей глибокого навчання.

Об’єкт дослідження – процеси обробки великих даних в

автоматизованих системах обробки інформації.

Предмет дослідження – методи та засоби інтеграції систем обробки великих даних та моделей глибокого навчання.

Висновки до розділу 1

1. Проведено дослідження сучасного стану задачі обробки великих даних, яке показало недосконаліть і обмеженість існуючих систем розподіленої обробки великих даних.

2. Проведене дослідження підтвердило актуальність в розробленні нових підходів для інтеграції розподілених систем обробки великих даних та моделей глибокого навчання.

2 МЕТОДИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ МОДЕЛЕЙ ГЛИБОКОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ

2.1 Нейромережеві моделі глибокого навчання для обробки великих даних

Методи глибокого навчання надали потужні інструменти для роботи з великими обсягами даних, оскільки вони витягують з них функції вищого рівня для отримання ієрархічних представлень.

Глибоке навчання було застосовано до галузей розпізнавання мовлення, акустичного моделювання для класифікації звуку, обробки зображень, таких як рукописна класифікація, класифікація сцен дистанційного зондування з високою роздільною здатністю, обробка природної мови, комп'ютерний зір, розпізнавання образів тощо [32].

Розглянемо існуючі дослідження в області глибоких нейронних мереж, які представлені в [22]. Як уже зазначалося, глибока нейронна мережа містить безліч прихованих шарів (рисунок 2.1) і здійснює глибоке ієрархічне перетворення вхідного простору образів.

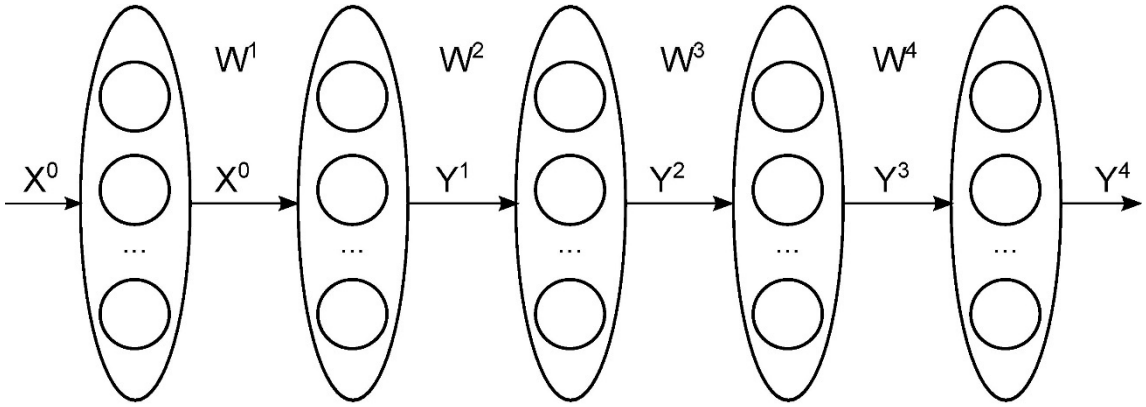

Рисунок 2.1 – Структура глибокої нейронної мережі

Вихідне значення j-го нейрона k-го шару визначається наступним чином:

$$y_j^k = F(S_j^k), \tag{2.1}$$

$$S_j^k = \sum_{i=1} w_{ij}^k y_i^{k-1} + T_j^k, \tag{2.2}$$

де F - функція активації нейронного елемента;

S_j^k - зважена сума j-го нейрона k-го шару;

w_{ij}^k - ваговий коефіцієнт між i-им нейроном (k-1)-го шару і j-м нейроном k-го шару;

T_j^k - порогове значення j-го нейрону k-го шару.

Для першого (розподільного) шару

$$y_i^0 = x_i. \tag{2.3}$$

У матричному вигляді вихідний вектор k-го шару

$$Y^k = F(S^k) = F(W^k Y^{k-1} + T^k), \tag{2.4}$$

де W - матриця вагових коефіцієнтів;

Y^{k-1} - вихідний вектор (k-1)-го шару;

T^k - вектор порогових значень нейронів k-го шару.

Якщо глибока нейронна мережа використовується для класифікації образів, то вихідні значення мережі часто визначаються на основі функції активації SoftMax:

$$y_j^F = softmax(S_j) = \frac{e^{S_j}}{\sum_l e^{S_l}}. \tag{2.5}$$

Попереднє навчання DBN виконується на основі обмеженої машини

Больцмана (RBM) або автоенкодера [22–47].

Розглянемо обмежену машину Больцмана, яка складається з двох шарів: видимого і прихованого (рисунок 1.2).

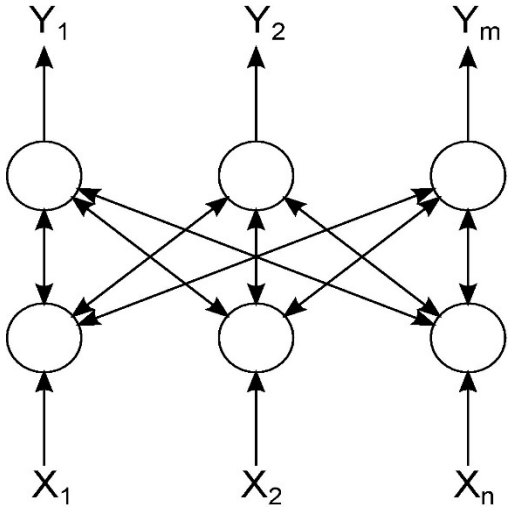

Рисунок 1.2 – Обмежена машина Больцмана

У RBM нейрони прихованого шару є детекторами ознак, які виділяють закономірності вхідних даних. Кожен нейрон має двосторонній зв’язок з іншими в сусідньому шарі. Обмежена машина Больцмана може генерувати дискретний розподіл, якщо використовується достатня кількість нейронів прихованого шару [8].

RBM є стохастичною нейронною мережею, в якій стани видимих і прихованих нейронів змінюються відповідно до ймовірнісної версії сигмоїдної функції активації:

$$p(y_j \mid x) = \frac{1}{1+e^{-S_j}}, S_j = \sum_i^n w_{ij} x_i + T_j \tag{2.6}$$

$$p(x_i \mid y) = \frac{1}{1+e^{-S_i}}, S_i = \sum_j^m w_{ij} y_j + T_i . \tag{2.7}$$

Необхідно відзначити, що стани видимих і прихованих нейронних елементів приймаються незалежними:

$$P(x \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$
$$P(y \mid x) = \prod_{j=1}^{m} P(y_j \mid x) \quad . \tag{2.8}$$

Використовуючи цей підхід, Хінтон [21] запропонував використовувати метод контрастної дивергенції (contrastive divergence, CD) для навчання RBM. У разі CD-1 правило навчання визначається як:

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \alpha(x_i(0)y_j(0) - x_i(1)y_j(1)) \\ T_i(t+1) &= T_i(t) + \alpha(x_i(0) - x_i(1)) \\ T_j(t+1) &= T_j(t) + \alpha(y_j(0) - y_j(1)). \end{aligned} \tag{2.9}$$

У випадку CD-k

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \alpha(x_i(0)y_j(0) - x_i(k)y_j(k)) \\ T_i(t+1) &= T_i(t) + \alpha(x_i(0) - x_i(k)) \\ T_j(t+1) &= T_j(t) + \alpha(y_j(0) - y_j(k)). \end{aligned} \tag{2.10}$$

У цьому випадку, перші доданки в правилах навчання характеризують розподіл даних в момент часу $t = 0$, а другі доданки характеризують реконструйовані або генеровані моделлю стани в момент часу $t = k$. Тут α - швидкість навчання. З останніх виразів видно, що правило навчання обмеженої машини Больцмана мінімізує різницю між оригінальними даними та даними, що генеруються моделлю. Дані, що генеруються моделлю, отримуються за допомогою семплінг-методу Гіббса.

Навчання RBM здійснюється наступним чином: представляється навчальний образ видимому шару нейронів, потім, використовуючи процедуру CD-n, обчислюються бінарні стани прихованих нейронів, виконується відновлення станів видимих нейронів і т.д. Після виконання цих ітерацій ваги і пороги обмеженої машини Больцмана модифікуються.

Метод навчання RBM грунтується на мінімізації середньоквадратичної

помилки реконструкції видимих і прихованих образів, яку можна отримати, використовуючи ітерації семплінг-методу Гіббса. У порівнянні з традиційним підходом, заснованим на мінімізації енергії (energy-based method), який базується на лінійному представленні нейронних елементів, запропонований метод дозволяє враховувати нелінійну природу нейронних елементів.

В наступному параграфі розглянемо методи реалізації моделей глибокого навчання для задач із великими даними.

2.2 Методи реалізації моделей глибокого навчання для обробки великих даних

2.2.1 Розподілене глибоке навчання для великих даних

Використання великих наборів даних на навчання архітектур нейронних мереж має великий вплив в багатьох областях, але потребує великих обчислювальних потужностей. Оскільки навчання нейронних мереж на одній машині часто займає дуже багато часу, з урахуванням досягнень у мережевих архітектурах, устаткуванні GPU та методах навчання, для розподіленого навчання нейронних мереж було проведено ряд досліджень. Наступні два підходи можуть бути розглянуті для розпаралелювання / розподілу навчання нейронних мереж [7]:

1. Паралелізм даних: дані розподіляються між вузлами.
2. Паралелізм моделі: параметри моделі розподіляються між вузлами.

2.2.1.1 Паралелізм даних

У розподіленому глибокому навчанні з паралелізмом даних навчальні дані поділяються на невеликі фрагменти даних і розподіляються по вузлах, а параметри моделі реплікуються в кожному вузлі, як показано на рисунку 2.1. Кожен вузол оновлює локальну для нього модель, проводячи навчання однієї невеликої партії локальних навчальних даних. Потім з оновлених моделей

(або градієнтів, які використовуються для оновлення моделі) створюється нова модель, яка передається всім вузлам.

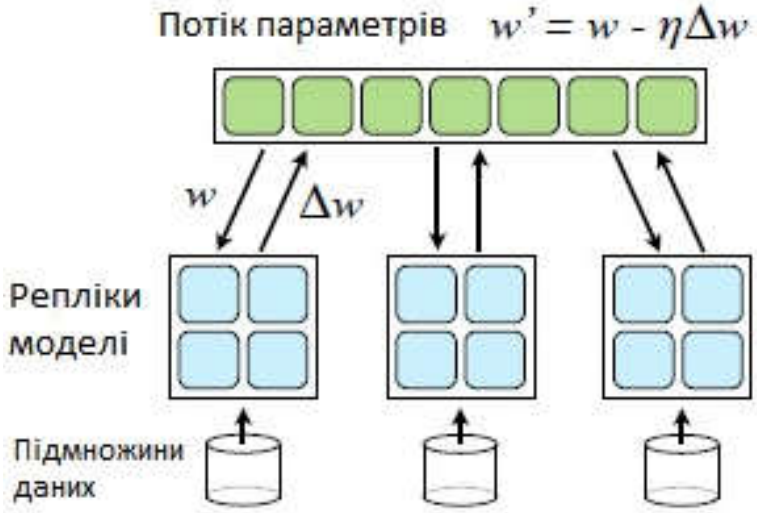

Рисунок 2.1 – Глибоке навчання з паралелізмом даних [12]

Мобільна аналітика великих даних із використанням Deep Learning та Apache Spark.

Автори в [2] пропонують підхід до виконання аналітики мобільних великих даних (MBD) з використанням Deep Learning та Apache Spark Framework. Запропонована структура має на меті прискорити процес прийняття рішень Mobile Big Data шляхом паралельного навчання та навчання моделей глибокого навчання на базі високопродуктивного обчислювального кластеру.

Запропонована структура використовує Apache Spark для вирішення проблеми обсягу, швидкості та різноманітності MBD. Модель глибокого навчання вирішує проблему різноманітності та цінності MBD.

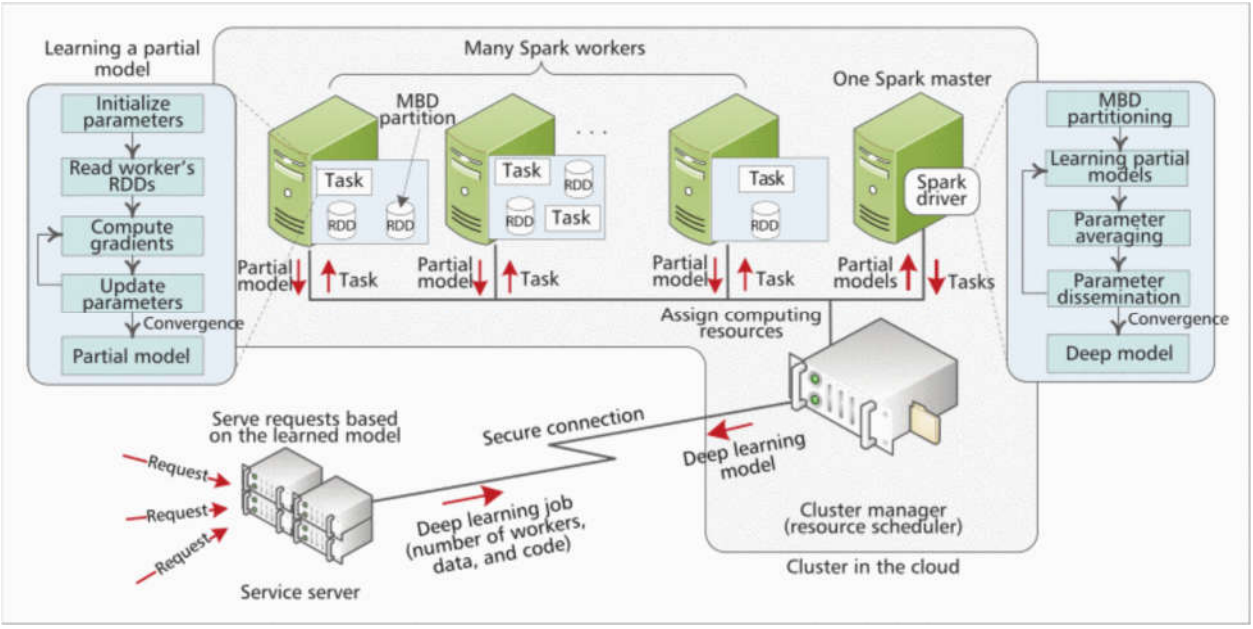

Рисунок 2.2 – Фреймворк на основі Spark розподіленого глибокого навчання для аналітики MBD [2]

При такому підході великі мобільні дані поділяються на невеликі пакети даних і розподіляються на кожному вузлі Spark Worker. Дані зберігаються у стійкому розподіленому наборі даних (Resilient Distributed Dataset, RDD) двигуна Spark. Модель глибокого навчання виконується на багатьох Spark Worker як ітеративні обчислення Map-Reduce. Кожен Spark Worker навчає свою відповідну локальну модель за допомогою розділу загальних мобільних великих даних.

Після усереднення параметрів усіх часткових моделей, отриманих від кожного Worker, будується головна глибока модель.

Фреймворк Spark складається із двох основних компонентів:

1. Spark Master: відповідає за ініціалізацію екземпляра драйвера Spark, який керує виконанням багатьох часткових моделей групи Spark Worker.

2. Одого або декілька Spark Worker: драйвер Spark, ініціалізований головною машиною, керує виконанням часткових моделей глибокого навчання у групі Spark Worker. Кожен робочий вузол Spark на кожній ітерації алгоритму глибокого навчання вивчає часткову глибоку модель на невеликих

розділах MBD і відправляє обчислені параметри на головний вузол. Головний вузол усереднює часткові моделі всіх вузлів-виконавців та відновлює основну модель.

Навчання глибоких моделей виконується у два етапи:

1. Обчислення градієнта.

2. Оновлення параметрів: оновлення параметрів складається з двох кроків. У першому етапі алгоритм навчання обчислює градієнти параметрів моделі локально шляхом незалежного перебору всіх розподілених пакетів даних. Другий крок обчислює середнє значення всіх параметрів моделі локального градієнта, отриманих на першому етапі, і відповідно оновлює параметри моделі навчання.

Ці два кроки описують модель глибокого навчання в моделі програмування Map-Reduce, в якій функція map виконує обчислення градієнта для всіх пакетів даних паралельно, а функція reduce оновлює параметри моделі глобально.

Платформа аналізу великих даних з використанням Apache Spark і глибокого навчання.

Автори роботи [19] запропононувдали підхід, якмй є кращим від традиційних методів аналізу даних, що використовують або Hadoop/Spark, або глибоке навчання в якості окремих елементів. Вони пропонують платформу, яка використовує розподільні обчислювальні можливості Spark та архітектуру глибокого навчання багатошарового перцептрона (MLP) з використанням каскадного навчання.

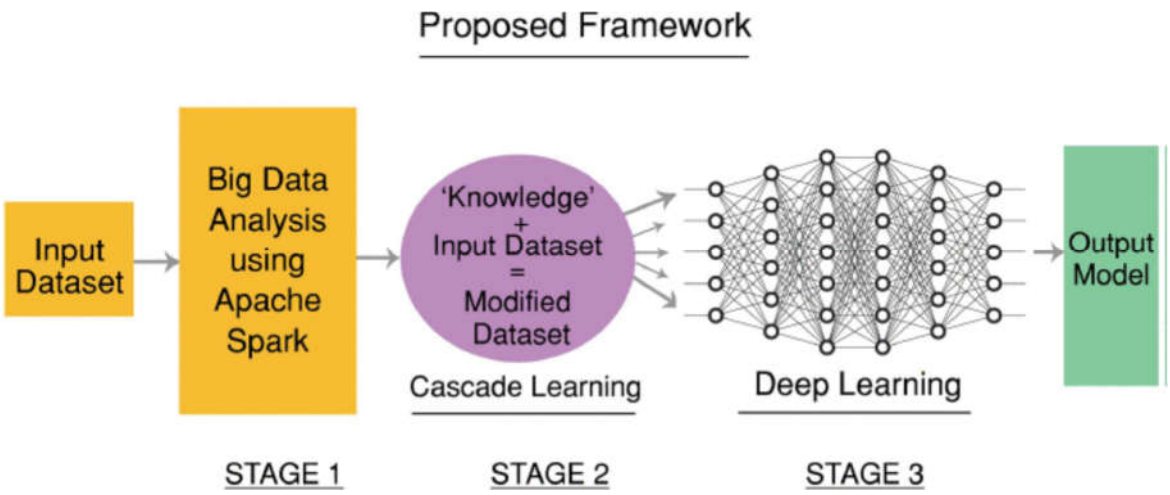

Рисунок 2.3 – Схематичне представлення фреймворка з використанням Apache Spark та Deep Learning [19]

Фреймворк складається з трьої етапів, що становлять основу досліджень і експериментів. Нижче описано ці три етапи:

Етап 1:

Цей етап є етапом бінарного навчання, на якому модель регресії, створена шляхом проходження попередньо обробленого набору даних через алгоритми регресії, представляє ймовірність належності кожної точки даних до бінарного класу. Цей етап в основному складається з Apache Spark, оскільки він використовує бібліотеку MLLib Spark для реалізації алгоритмів регресії, таких як дерево рішень, випадкові ліси та логістична регресія.

Етап 2:

Етап 2 - це етап каскадного навчання. Процес каскадування включає використання результатів, які ми отримуємо від однієї моделі, для навчання іншої моделі. У цій структурі етап 2 змінює вихідний набір даних, додаючи ймовірності, отримані на етапі 1. Імовірності виявилися сильною відмінною рисою в модифікованому наборі даних і використовувалися в якості значення базової істини для кожної точки даних. Цей модифікований набір даних використовується в якості вхідних даних для етапу 3.

Етап 3:

На цьому етапі використовується модифікований набір даних, отриманий на етапі 2, для навчання архітектури багатошарового персептрона (MLP). Архітектура MLP змінюється в залежності від розглянутої програми. Відповідно до вимог програми, цей етап можна використовувати для бінарної або багатокласової класифікації.

Глибина мережевої архітектури MLP залежить від складності задачі та складності обчислень системи.

Окрім переваг спільного використання Spark та моделей глибокого навчання, цей підхід також пропонує інші важливі переваги, такі як розширений набір функцій, отриманий на етапі 2, покращує загальну точність моделі. Крім того, використання бібліотеки MLLib Spark і моделі глибокого навчання разом зменшило необхідний час обчислень у порівнянні з використанням двошарового підходу з використанням двох моделей глибокого навчання одночасно. Фреймворк використовує модель глибокого навчання на основі зворотного поширення, що призводить до безперервного навчання та вдосконалення. Отже, це робить фреймворк більш надійним.

Однією з проблем цього підходу, є те, що коли обидва етапи є мультикласовими класифікаторами, фреймворк дає нижчу продуктивність. Експерименти показали, що використання багатокласового класифікатора етапу 1 у поєднанні з багатокласовою моделлю глибокого навчання на етапі 3 призводило до нижчої продуктивності порівняно з тим, коли обидва етапи використовували двійкові класифікатори.

2.2.1.2 Паралелізм моделі

У розподіленому глибокому навчанні з паралелізмом моделі параметри моделі поділяються і призначаються кожному обчислювальному вузлу, а дані для навчання моделей реплікуються на кожному вузлі, як показано на рисунку 2.4. Інформація, зібрана з моделі кожного іншого вузла, оновлює локальну модель.

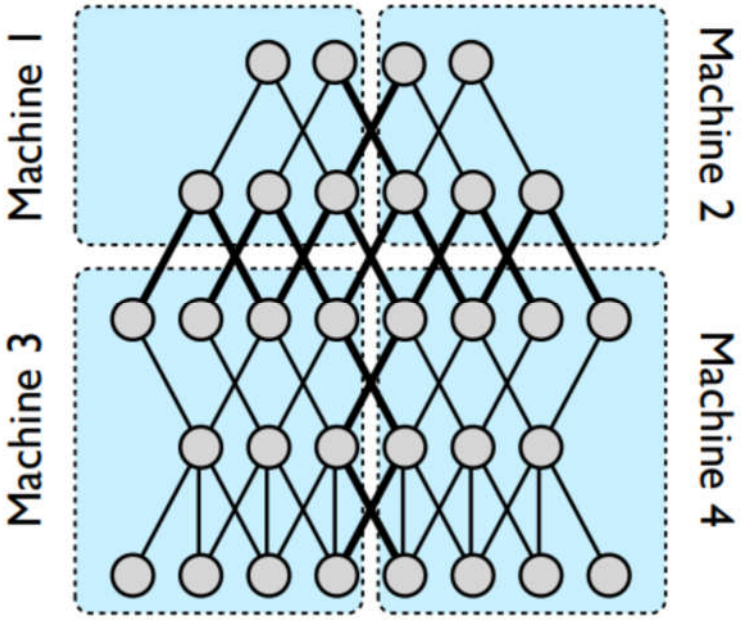

Рисунок 2.4 – Паралелізм моделі [12]

Розпаралелювання моделей і стратегії планування для розподіленого машинного навчання.

Автори [31] розробили систему для паралелізму моделей – Structure Aware Dynamic Scheduler (STRADS), яка виконує автоматичне планування, а також пріоритизацію параметрів для реалізації паралелізму моделей.

Щоб використати паралелізм моделей, STRADS надає програмний інтерфейс, в якому користувачі можуть написати три функції для будь-якого завдання машинного навчання: schedule, push та pull. STRADS створює алгоритм ітеративного паралелізму моделей, повторюючи планування та виконуючи ці три функції.

Розглянемо спис цих трьох функцій:

Schedule: ця функція вибирає параметри моделі, які будуть надіслані для оновлення. Вона вибирає параметри моделі рівномірно або у фіксованій послідовності. При виборі параметрів моделі вона гарантує, що вибираються тільки параметри, що найбільш збігаються, щоб уникнути вже збіжних параметрів, і дозволяє уникнути помилок розпаралелювання, не відправляючи параметри з взаємозалежностями паралельно.

Push and Pull: функції Push та Pull описують потік параметрів моделі від планувальника до worker. Функції push надсилають вибрані параметри моделі із планувальника кожному виконавцю. Потім кожен виконавець обчислить часткове оновлення параметрів моделі. Функція Push використовується для збирання часткових оновлень від усіх робочих процесів та їхньої фіксації для повного оновлення параметрів.

STRADS використовує кілька головних/машин-планувальників, робочих машин і одну головну машину, яка координує потік між планувальниками та виконавцями. STRADS виконує функції schedule-push-pull в циклічному порядку.

Етапи реалізації показані на рисунку 2.5.

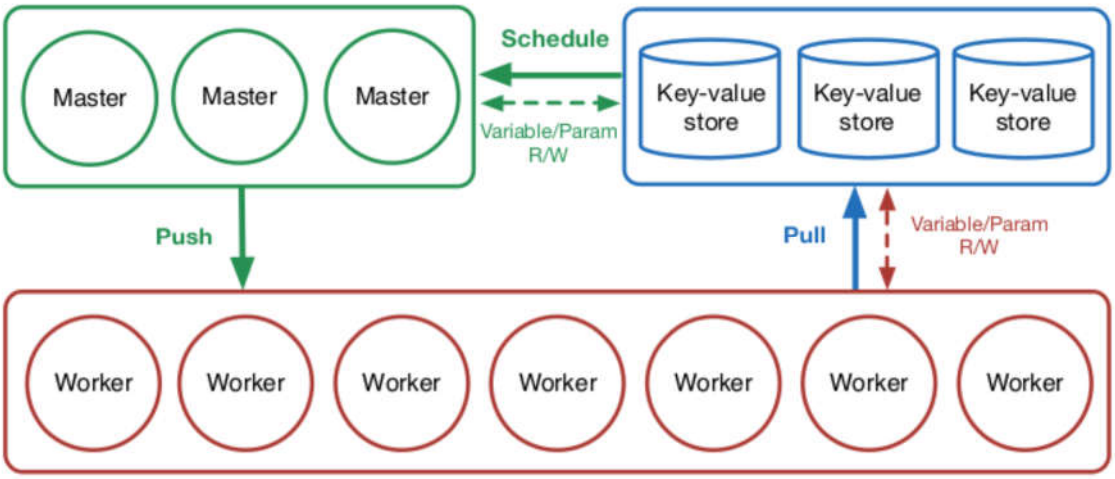

Рисунок 3.5 – Високорівнева архітектура інтерфейсу системи STRADS для паралелізму динамічної моделі [31]

1) головна машина виконує функцію schedule, щоб вибрати параметри моделі, які будуть використовуватися паралельно;

2) функція push виконується для відправлення параметрів моделі на робочі машини та обчислення часткових оновлень для кожного параметра;

3) сховища ключів і значень виконують функцію витягування, щоб агрегувати часткові оновлення та зберігати нещодавно оновлені параметри.

STRADS забезпечує масштабованість та ефективне використання пам’яті, дозволяючи більшим моделям працювати з додатковими машинами.

Він також надає можливість викликати динамічні розклади, які зменшують залежності параметрів моделі між виконавцями, що призводить до меншої похибки розпаралелювання і, отже, швидшої і правильної збіжності.

2.2.1.3 Паралелізм моделі даних

Паралелізм моделей і даних використовує концепцію централізованого сервера параметрів, який відповідає за зберігання та оновлення параметрів моделі. Кожен вузол містить розділ навчальних даних і репліку моделі нейронної мережі, як показано на рисунку 2.6. Кожен вузол самостійно виконує навчання однієї партії локальних навчальних даних і оновлює локальну модель. Потім він надсилає оновлення моделі на сервер параметрів. Кожен вузол отримує оновлення моделі інших вузлів від сервера параметрів.

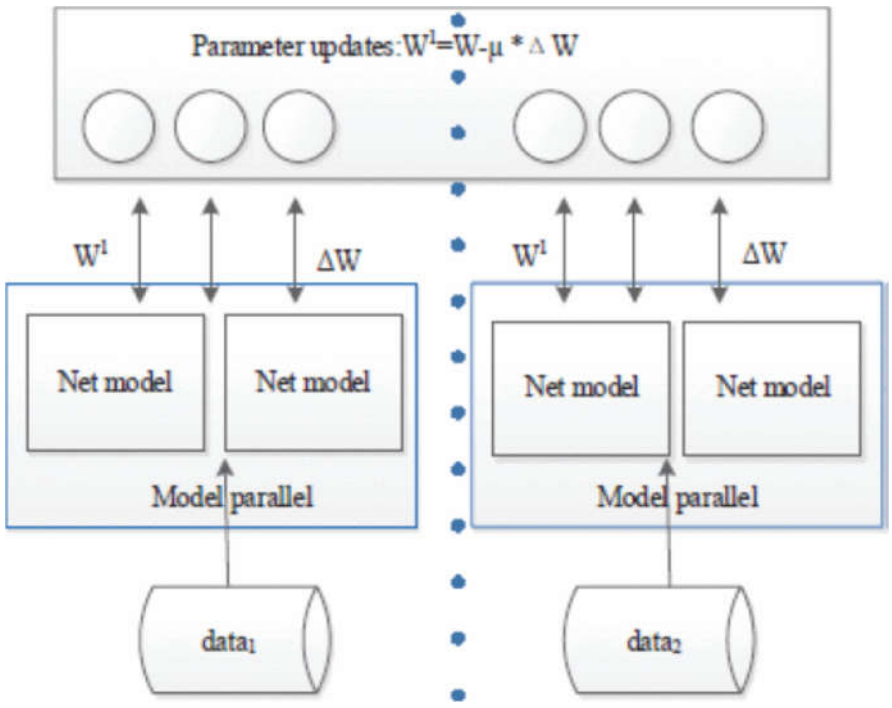

Рисунок 2.6 – Паралелізм моделі даних

Розподілена платформа глибокого навчання на основі Spark для додатків з великими даними представлена на рисунку 2.7.

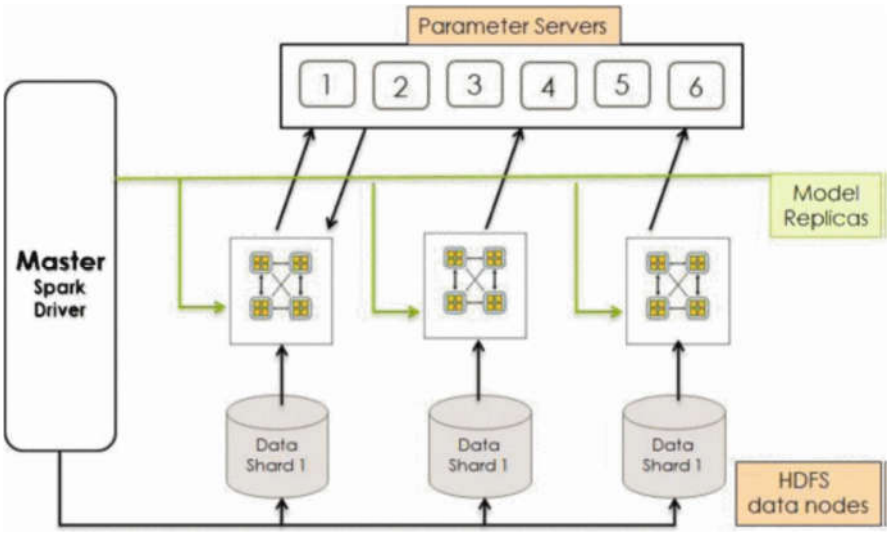

Рисунок 2.7 –Загальна архітектура розподіленої системи глибокого навчання на основі Spark для додатків великих даних [26]

Автори в [26] розробили та впровадили фреймворк для навчання моделей глибокого навчання за допомогою Apache Spark. Цей фреймворк прискорює час навчання великомасштабних мереж глибокого навчання, розподіляючи ту саму модель за допомогою стохастичного градієнтного спуску серед кластерів вузлів даних у HDFS.

Цей фреймворк використовує переваги як паралелізму даних, так і моделі. Розподіл навчальних даних на машинах кластера Spark і реплікація моделі на кожній машині реалізує паралелізм даних. Кожна модель тренується паралельно зі своїм розділом даних. Розподіл шарів кожної репліки моделі глибокої нейронної мережі по кластеру Spark реалізує паралелізм моделі.

Фреймворк складається з трьох основних компонентів:

Spark Master: він має два основних компоненти, а саме Spark Driver, який відповідає за обслуговування, координацію та планування додатків у

кластері, і робочі вузли (Worker Nodes), які відповідають за запуск програм. Він починає процес навчання нейронної мережі з ініціалізації параметрів Parameter Server і рівнях нейронної мережі.

Parameter Server: сервер параметрів розділений між машинами в кластері Spark, і кожен розділ відповідає шару моделі нейронної мережі.

Data Shard: сегменти даних – це вузли даних. Відповідно до розміру мережевого рівня кожен фрагмент даних генерує шари нейронної мережі, і для кожного розділу даних він обчислює прямі та зворотні проходи для кожного прикладу даних, поки не обробить усі приклади. Він оновлює головну модель після кожної ітерації, надсилаючи нещодавно обчислені градієнти до головної моделі Parameter Server.

2.2.2 Широкомасштабне глибоке навчання для великих даних

Хоча глибоке навчання показало чудові результати у багатьох додатках, таке навчання нелегке для додатків великих даних, оскільки воно потребує великих обчислювальних ресурсів. Ці обчислення ускладнюються зі зростанням наборів даних. Таким чином, з безпрецедентним зростанням комерційних наборів даних в останні роки, спостерігається сплеск інтересу до ефективних і масштабованих паралельних алгоритмів для навчання глибоких моделей. У цьому параграфі розглянемо високопродуктивні обчислювальні пристрої та архітектури, такі як графічні процесори (GPU) і кластери CPU, які дають змогу навчати великі моделі глибоких нейронних мереж для вивчення функцій великих даних.

2.2.2.1 Глибоке навчання з використанням ЦП

Великі розподілені глибокі мережі.

Dean та ін. [12] розробили програмний фреймворк Distbelief, який може використовувати обчислювальну потужність кластерів із тисячами машин для навчання великомасштабних алгоритмів глибокого навчання. Фреймворк Distbelief забезпечує паралелізм моделі всередині вузла за допомогою

багатопотокової передачі та через вузол за допомогою передачі повідомлень [26]. У рамках цього розроблено два алгоритми для широкомасштабного навчання алгоритмів глибокого навчання:

1. Стохастичний градієнтний спуск (Downpour Stochastic Gradient Descent, SGD): асинхронна процедура стохастичного градієнтного спуску, що підтримує велику кількість реплік моделі з адаптивною швидкістю навчання.

2. Sandblaster: фреймворк, який підтримує розподілені процедури пакетної оптимізації, включаючи розподілену реалізацію L-BFGS (Limited-Memory Broyden Fletcher Goldfarb Shanno) [35], яка використовує паралелізм як моделі, так і даних.

Обидва алгоритми спроектовані таким чином, що вони можуть допускати зміну швидкості виконання реплік різних моделей і стійкіші до збоїв машини, таких як відключення машини або випадковий перезапуск. Обидва оптимізовані алгоритми реалізують інтелектуальну версію паралелізму даних. Вони також дозволяють одночасно виконувати окремі навчальні приклади в кожній репліці моделі та періодично збирати їх результати для оптимізації цільової функції моделі.

DistBelief отримав високу швидкість для навчання кількох великомасштабних моделей глибокого навчання. Наприклад, він прискорився в 12 разів для згорткової нейронної мережі з 1,7 мільярда параметрів і 16 мільйонами зображень на 81 машині.

Крім того, він також досяг значного покращення для навчання іншої архітектури глибокого навчання з 14 мільйонами зображень розміром 200x200 пікселів на 1000 машинах з 16 ядрами ЦП. Фреймворк Distbelief надзвичайно ефективний для навчання моделей глибокого навчання для вивчення функцій великих даних, оскільки його можна масштабувати на багатьох комп'ютерах.

2.2.2.2 Широкомасштабне глибоке навчання з використанням графічних процесорів

Сьогодні сучасні ПК комплектуються типовою графічною картою, яка містить понад сотню процесорних ядер і має пропускну здатність пам’яті в кілька разів більше, ніж їх центральні процесори. Апаратне забезпечення призначене для одночасної роботи з тисячами потоків з дуже невеликими обчислювальними витратами. Це робить графічні процесори все більш привабливими для обчислень загального призначення, які важко розпаралелити на інших розподілених архітектурах.

Деякі експерименти були проведені для оцінки широкомасштабних фреймворків глибокого навчання, таких як Caffe, CNTK1 (Microsoft Cognitive toolkit), MXNet2 і Tensorflow, в середовищі з одним і кількома графічними процесорами [40].

На рисунку 2.8 показано спрощену схему типового графічного процесора Nvidia. Типовий GPU складається з кількох мультипроцесорів (MP), і кожен мультипроцесор містить кілька потокових процесорів (SP), які відповідають за фактичні обчислення. Кожен MP запускає заплановану групу потоків, які називаються блоками. У кожному MP кожен блок планується виконувати на SP. Кожен графічний процесор має глобальну пам’ять з дуже високою пропускною здатністю та високою затримкою під час доступу зі сторони ЦП (хоста).

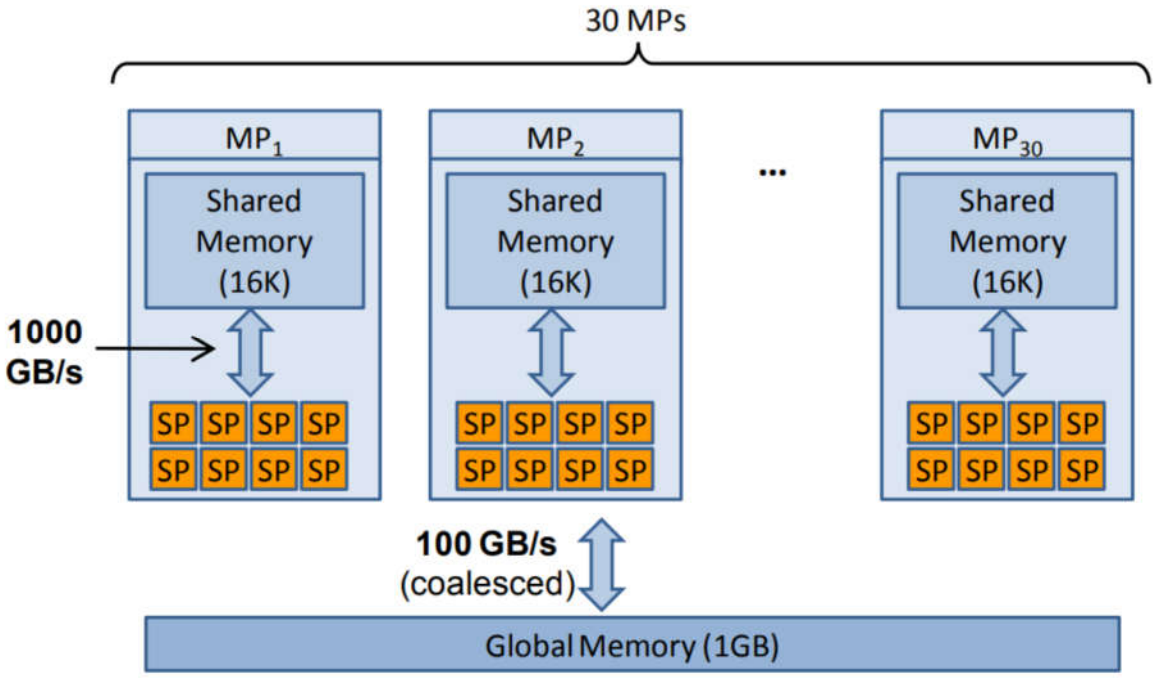

Рисунок 2.8 – Узагальнена структурна схема відеокарти Nvidia GeForce GTX 280 [38]

Апаратне забезпечення графічного процесора допускає два рівні паралелізму: рівень інструкцій (пам'яті) (тобто MP) і рівень потоку (SP). Усі потоки в блоці мають спільний доступ до невеликого обсягу локальної спільної пам'яті (16 КБ). Вони також мають доступ до глобальної пам'яті GPU.

Широкомаштабне глибоке навчання без вчителя за допомогою графічних процесорів.

Raina та ін. [38] пропонують фреймворк на основі графічного процесора для паралелізації моделей навчання без вчителя, включаючи DBN і розріджене кодування. Основною метою було навчити неконтрольовані моделі навчання для великомасштабних додатків з мільйонами вільних параметрів, що змушує дослідників використовувати менш масштабні моделі або скорочувати навчальні приклади.

Оскільки передача даних між хостом і глобальною пам'яттю графічного процесора займає багато часу, існує необхідність мінімізувати передачу між хостом і пристроєм, використовуючи переваги спільної пам'яті.

Однією зі стратегій скорочення часу передачі є збереження всіх параметрів і великої частини навчальних прикладів у глобальній пам’яті під час навчання. Ця стратегія дозволяє повністю оновлювати параметри всередині графічних процесорів. Щоб скористатися перевагами рівнів паралелізму MP/SP, як показано на рисунку 2.8, на кожній ітерації буде вибрано кілька навчальних прикладів із глобальної пам’яті, щоб одночасно обчислювати оновлення між блоками, що допомагає досягти паралелізації даних.

Експериментальні результати показують, що з 45 мільйонами параметрів у RBM та мільйоном прикладів, реалізація на основі GPU збільшує швидкість навчання DBN до 70 разів у порівнянні з двоядерною реалізацією CPU [38].

2.3 Метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання

Запропонований метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання, на відміну від відомих рішень, передбачає:

– можливість обробляти різні типи даних і формати даних;

– можливість одночасної обробки даних з машини Hadoop або Spark;

– можливість інтеграції розподілених систем обробки великих даних з моделями глибокого навчання;

– можливість вирішення не лише певних задач, таких як класифікація тексту або класифікація зображень, а також аналіз потокових даних;

– можливість змінювати значення параметрів навчання моделі перед виконанням різноманітних експериментів.

Запропонований метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання складається з наступної послідовності кроків:

1. Користувач вибирає набір даних для проведення експериментальних досліджень з обробки великих даних. Це можуть бути пакетні дані від Apache Hadoop та потоки даних у режимі реального часу від Apache Spark чи Apache Storm. Також є можливість вибрати та завантажити JSON-файл для проведення експериментальних досліджень. Тобто, користувач має можливість вибрати то чи інше джерело великих даних.

2. Користувач має пройти автентифікацію у випадку використання Hadoop чи іншої розподіленої системи обробки великих даних. Для цього необхідно ввести пароль або SSH-ключ для доступу. Після успішного доступу користувач може переглянути список каталогів та файлів на віддаленій машині, а також може переглянути вміст того чи іншого файлу і вибрати відповідний для подальшої роботи.

3. Користувач може вибрати операції, які необхідно провести на етапі попередньої обробки даних. Такі операції вибираються в залежності від типу даних. Це можуть бути наступні операції для числових даних:

– очищення даних – видалення зашумлених, надлишкових, невідоповідних даних;

– інтеграція даних – об’єднання значень атрибутів даних з різних джерел;

– стиснення даних – зменшення розміру даних на основі відповідних методів;

– перетворення даних – може включати нормалізацію, агрегацію та узагальнення даних.

4. Для графічних даних, тобто зображень, користувач може використати наступні операції: обрізання та обертання зображення, регулювання яскравості, насиченості та ін.

5. Користувач може вибрати тип задачі, яку необхідно вирішити (наприклад, класифікація, прогнозування та ін.).

6. Користувач може вибрати тип нейронної мережі для вирішення тої чи іншої задачі з бази нейронних мереж, підготовлених для їх вирішення. На

цьому кроці користувач має можливість налаштовувати параметри навчання нейронних мереж та проводити безпосередньо аналіз даних.

Узагальнений алгоритм роботи методу в режимі обробки великих даних представлено на рисунку 2.9.

Рисунок 2.9 – Узагальнений алгоритм роботи методу в режимі обробки великих даних

На рисунку 2.10 представлено узагальнений алгоритм роботи методу в режимі навчання нейромережевих моделей.

Рисунок 2.10 – Узагальнений алгоритм роботи методу в режимі навчання нейромережевих моделей

Висновки до розділу 2

1. Проведено дослідження методів реалізації моделей глибокого навчання для обробки великих даних. Використання великих наборів даних на навчання архітектур нейронних мереж має великий вплив в багатьох областях, але потребує великих обчислювальних ресурсів. Сучасні досягнення мережевих архітектур, GPU та методів навчання дозволяють вирішити такі проблеми.

2. Вдосконалено метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання, що дозволить трансформувати дані, що надходять з різних потоків великих даних, у формат, необхідний для навчання глибоких нейронних мереж, а також дозволить виконувати різні маніпуляції з потоками даних.

3. Розроблено узагальнений алгоритм роботи методу в режимі обробки великих даних та узагальнений алгоритм роботи методу в режимі навчання нейромережевих моделей.

3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДУ ІНТЕГРАЦІЇ РОЗПОДІЛЕНИХ СИСТЕМ ОБРОБКИ ВЕЛИКИХ ДАНИХ З МОДЕЛЯМИ ГЛИБОКОГО НАВЧАННЯ

3.1 Інструментальні засоби попередньої обробки великих даних

Для інтеграції потоків великих даних з моделями глибокого навчання потрібна попередня обробка даних, щоб отримати дані у належному форматі. Великі дані можуть бути непослідовними, неповними, зайвими, ненадійними, зашумленими та неактуальними для подальшого аналізу. Успіх будь-якої моделі глибокого навчання залежить від якості даних, які використовуються для аналізу.

Результатом попередньої обробки є набір даних, який можна використовувати як остаточний навчальний набір для алгоритмів машинного навчання [29]. Сьогодні існує багато інструментів попередньої обробки даних.

WEKA (Waikato Environment for Knowledge Analysis) – це платформо-незалежна система, яка надає повний набір бібліотек Java, що підтримують кілька стандартних задач інтелектуального аналізу даних, включаючи попередню обробку даних, кластеризацію, класифікацію, регресію, візуалізацію та вибір функцій [44].

WEKA надає можливості попередньої обробки даних у вигляді фільтрів, які дозволяють обробляти дані на рівні екземплярів та значень атрибутів. Нижче наведено фільтри, які WEKA використовує для атрибутів набору даних [46]:

1. Загальні маніпуляції з атрибутами: Нижче розглянемо список фільтрів для маніпуляцій з атрибутами:

– додання/видалення фільтру: щоб вставити та видалити атрибути;

– створення індикаторного фільтру: він перетворює номінальний атрибут на бінарний індикаторний атрибут. Він використовується, коли

атрибут з кількома класами має бути представлений атрибутом із двома класами;

– фільтр «об'єднання значень атрибутів»: він поєднує значення атрибутів в одне значення;

– фільтр перетворення номінальних значень на двійкові: він перетворює багатозначні номінальні атрибути на двійкові атрибути;

– вибір фільтру: видаляє всі екземпляри з набору даних, які показують одне з певного набору номінальних значень атрибутів нижче або вище за певний поріг;

– фільтр «замінення відсутніх значень»: кожне відсутнє значення замінюється середнім (для числових атрибутів);

– фільтр «змінити атрибути місцями»: змінює значення атрибутів місцями.

2. Перетворення числових атрибутів: деякі фільтри попередньої обробки даних призначені спеціально для числових атрибутів, наприклад:

– фільтр числового перетворення: він перетворює всі числові атрибути за допомогою заданої користувачем функції перетворення;

– дискретний фільтр: реалізує контрольований та неконтрольований метод дискретизації.

3. Вибір функцій: він надає три способи вибору функцій, тобто локально створений метод на основі кореляції, метод оболонки та полегшення для вибору відповідних атрибутів, які будуть включені в індукцію моделі.

Автори в [28] описують паралельний розподілений фреймворк Weka для аналізу великих даних за допомогою Spark під назвою DistributedWekaSpark. Фреймворк спрямований на подолання недоліків Weka, оскільки він підтримує лише послідовне виконання на одному вузлі, що накладає значні обмеження в обробці великих даних.

Цей фреймворк є масштабованим набором інструментів для інтелектуального аналізу великих даних, який поєднує в собі обчислювальну

потужність Spark зі стандартною зручністю використання Weka. Він використовує як паралельне, і розподілене виконання, оскільки побудований на основі Spark, який забезпечує швидку ітеративну обробку в пам'яті.

RapidMiner [39] – це програмна платформа, розроблена компанією Rapid Miner та забезпечує інтегроване середовище для підготовки даних, машинного навчання, інтелектуального аналізу тексту та прогнозного аналізу. RapidMiner розроблений Ральфом Клінкенбергом, Інго Мерсва та Саймоном Фішером з відділу штучного інтелекту Технічного університету Дортмунда у 2001 році і спочатку називався як Yet Another Learning Environment (YALE).

Rapid Miner – це кроссплатформне програмне забезпечення, що використовує модель клієнт/сервер, при цьому сервер пропонується як програмне забезпечення або у послуга хмарних інфраструктурах. Він підтримує двадцять два формати файлів і містить понад 100 навчальних схем для регресійного, класифікаційного та кластерного аналізу [13].

Radoop [37] – це розширення інструменту інтелектуального аналізу даних RapidMiner, який надає прості у використанні оператори для запуску розподілених процесів в Hadoop. RapidMiner надає користувачеві графічний інтерфейс для виконання таких операцій, як завантаження, видобуток корисних даних та задачі візуалізації. Radoop зчитує дані HDFS і виконує різні функції інтелектуального аналізу даних. Оператори, що використовуються в Radoop, аналогічні до Apache Mahout [4], бібліотеці для алгоритмів машинного навчання в Hadoop, і, отже, обидва мають однакові проблеми з продуктивністю. Під час виконання оператори переводяться в задачі Mahout та виконуються в Hadoop.

Radoop використовує HIVE [25] для виконання всіх перетворень даних, які виконуються як завдання Map-Reduce у Hadoop. Усі перетворення даних виражаються у сценаріях HiveQL. Він включає вибір атрибутів, створення нових атрибутів, повторення прикладів, сортування, перейменування,

перетворення типів і т. д. Він також підтримує агрегування, об'єднання таблиць і користувацькі функції (UDF) для більш складних перетворень [36].

Бертольд та ін. [10] розробили модульне середовище під назвою KNIME [27] (Konstanz Information Miner), яке забезпечує просте візуальне складання та інтерактивне виконання конвеєра даних. KNIME розроблений як платформа для навчання, досліджень та спільної роботи, яка полегшує просту інтеграцію нових алгоритмів та інструментів, а також методів обробки даних або візуалізації у вигляді нових модулів чи вузлів.

KNIME написаний з використанням JAVA, а його графічний редактор реалізований у вигляді модуля Eclipse. Архітектура KNIME була розроблена з використанням трьох принципів:

1. Візуальна інтерактивна структура: Drag&Drop дозволяє комбінувати потоки даних від різних пристроїв обробки.

2. Модульність: блоки обробки та контейнери даних не повинні залежати один від одного, щоб забезпечити простий розподіл обчислень та розробку різних алгоритмів.

3. Легка розширюваність: має бути легко додавати нові вузли обробки або представлення та поширювати їх за допомогою простого механізму плагінів без потреби у складних процедурах встановлення/видалення.

KNIME також має текстовий плагін [41] для обробки тексту та даних природною мовою. Це комбінація обробки природної мови, інтелектуального аналізу тексту та пошуку інформації. Використовуючи цей плагін, можна читати текстові дані, представляти дані всередині як документи та терміни та застосовувати до них кілька тегів, фільтрації, обчислення частоти, перетворення структури даних та інших завдань попередньої обробки та аналізу. Це проміжний плагін, який дозволяє читати, попередньо обробляти та перетворювати текстові дані у числові представлення.

3.2 Реалізація програмного забезпечення для інтеграції великих наборів даних та моделей глибокого навчання

Для реалізації запропонованого методу інтеграції великих наборів даних з моделями глибокого навчання розроблено програмне забезпечення мові на Python [50, 52].

Python – кроссплатформна мова програмування, що означає, що вона може працювати на кількох платформах, таких як Windows, macOS, Linux, і навіть була перенесена на віртуальні машини Java та .NET. Мова програмування Python безкоштовна і з відкритим кодом [50, 52].

Python – популярна мова програмування загального призначення, яку можна використовувати для широкого спектру додатків. Python включає високорівневі структури даних, динамічну типізацію, динамічне зв'язування і багато інших функцій, які роблять його настільки ж корисним для розробки складних додатків, як і для написання сценаріїв або «сполучного коду», що з'єднує компоненти разом. Його також можна розширити для виконання системних викликів майже всіх операційних систем та для виконання коду, написаного на C або C++. Завдяки своїй повсюдності та здатності працювати практично на будь-якій системній архітектурі, Python є універсальною мовою, яку можна знайти у багатьох додатках [50, 52].

Tkinter – це пакет Python, призначений для роботи з бібліотекою Tk. Бібліотека Tk містить компоненти графічного інтерфейсу користувача (graphical user interface – GUI). Ця бібліотека написана мовою програмування Tcl.

Код для реалізація кнопки:

```
login_btn = PhotoImage(file='butun/456.png')
img_label = Label(image=login_btn)
my_button =Button(root, image=login_btn, borderwidth=0,
pady=15, bg='#FFFFFF', activebackground='#FFFFFF' ).place(x = 329,y = 255)
my_label1 = Label(root, text='SELECT DATA STREAM', pady=2,
    font="Arial 32",  fg='#2699FB', bg='#FFFFFF' ).place(x = 236,y = 501 )
```

Для того щоб кнопка мала функцію завантаження файлу потрібно використати функцію filedialog з вказанням розширення завантажуваного файлу

```
root.filename =  filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("jpeg files","*.jpg"),("all files","*.*")))
```

Після того як було завантажено файли і вибрано усі потрібні умови. Відображається в новому вікні таблиця з даними. Реалізовано дві функції для додавання нових рядків і видалення рядків

Функція додавання рядків:

```
def add_row():
    global i
    i=i+1
    items = []
    var = IntVar()
    c = Checkbutton(root, variable = var)
    c.val = var
    items.append(c)
    c.grid(row = i, column = 0)
    for j in range(1,5): #Columns
        b = Entry(root)
        items.append(b)
        b.grid(row=i, column=j)
    rows.append(items)
```

Функція видалення рядків

```
def delete_row():
    for rowno, row in reversed(list(enumerate(rows))):
        if row[0].val.get() == 1:
            for i in row:
                i.destroy()
            rows.pop(rowno)
```

Для відображення графіків використовується бібліотека matplotlib. matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірною 2D графікою (3D графіка також підтримується). Отримувані зображення можуть бути використані як ілюстрації в публікаціях

Наступна функція створює і виводить графік для наших даних з потрібними умовами.

```
        def plot():
            fig = Figure(figsize = (5, 5), dpi = 100)
            y = [i**2 for i in range(101)]
            plot1 = fig.add_subplot(111)
            plot1.plot(y)
            canvas = FigureCanvasTkAgg(fig, master = window)
            canvas.draw()
            canvas.get_tk_widget().pack()
            toolbar = NavigationToolbar2Tk(canvas, window)
            toolbar.update()
            canvas.get_tk_widget().pack()
        window = Tk()
        window.title('Plotting in Tkinter')
        window.geometry("500x500")
        plot_button = Button(master = window, command = plot, height = 2, width = 10,
  text = "Plot")
        # in main window
        plot_button.pack()
        window.mainloop()
```

Також однією з цікавих частин коду є створення древа каталогів. Древа каталогів - елемент файлової системи, призначений для організації ієрархії файлової системи обчислювального пристрою шляхом групування файлів та інших каталогів.

```
      class App(object):
          def __init__(self, master, path):
              self.nodes = dict()
              frame = tk.Frame(master)
              self.tree = ttk.Treeview(frame)
              ysb = ttk.Scrollbar(frame, orient='vertical', command=self.tree.yview)
              xsb         =         ttk.Scrollbar(frame,         orient='horizontal',
command=self.tree.xview)
              self.tree.configure(yscroll=ysb.set, xscroll=xsb.set)
              self.tree.heading('#0', text='Project tree', anchor='w')
              self.tree.grid()
              ysb.grid(row=0, column=1, sticky='ns')
              xsb.grid(row=1, column=0, sticky='ew')
              frame.grid()
              abspath = os.path.abspath(path)
              self.insert_node('', abspath, abspath)
              self.tree.bind('<<TreeviewOpen>>', self.open_node)
          def insert_node(self, parent, text, abspath):
              node = self.tree.insert(parent, 'end', text=text, open=False)
              if os.path.isdir(abspath):
                  self.nodes[node] = abspath
                  self.tree.insert(node, 'end')
          def open_node(self, event):
              node = self.tree.focus()
```

```
abspath = self.nodes.pop(node, None)
                if abspath:
                    self.tree.delete(self.tree.get_children(node))
                    for p in os.listdir(abspath):
                        self.insert_node(node, p, os.path.join(abspath, p))
        if __name__ == '__main__':
            root = tk.Tk()
            app = App(root, path='.')
            root.mainloop()
        def main():
            yarn = Yarn("http://localhost:8088", 'json')
```

Для під’єднання до API Hadoop використовується така бібліотека Hadoop YARN REST Api. Код зображений дальше відповідає за під’єднання.

```
            response_obj = yarn.cluster_information()
            response_obj = yarn.cluster_metrics()
            response_obj = yarn.cluster_scheduler()
            response_obj = yarn.cluster_applications()
            response_obj = yarn.cluster_applications({"limit":100})
            response_obj = yarn.cluster_appstatistics()
            response_obj                                                    =
yarn.cluster_appstatistics({"states":"accepted,running,finished","applicationTypes":"
mapreduce"})
            response_obj = yarn.cluster_application("job_id")
            response_obj = yarn.cluster_application_attempts("job_id")
            response_obj = yarn.cluster_nodes()
            response_obj = yarn.cluster_nodes({"states":"RUNNING"})
            response_obj = yarn.cluster_node("node_id")
            if yarn.response_type == 'json':
                print(json.dumps(response_obj, indent=4, sort_keys=True))
            elif yarn.response_type == 'xml':

print(xml.dom.minidom.parseString(ET.tostring(response_obj)).toprettyxml())
            else:
                print(response_obj)
        if __name__ == '__main__':
            main()
```

Код програмного забезпечення подано в додатку А.

Схема взаємозв’язків програмних модулів реалізації запропонованого методу представлено на рисунку 3.1.

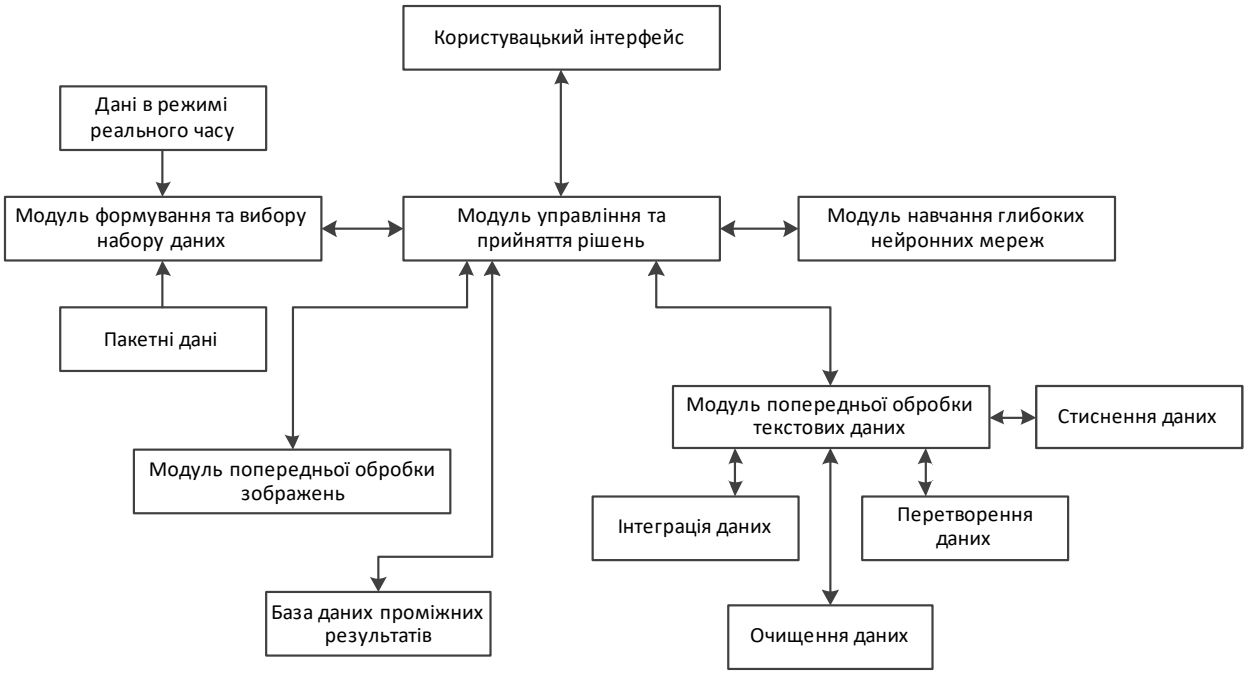

Рисунок 3.1 – Схема взаємозв’язків програмних модулів

Тепер розглянемо інтерфейс користувача.

Вікно вибору даних для проведення експерименту представлено на рисунку 3.2.

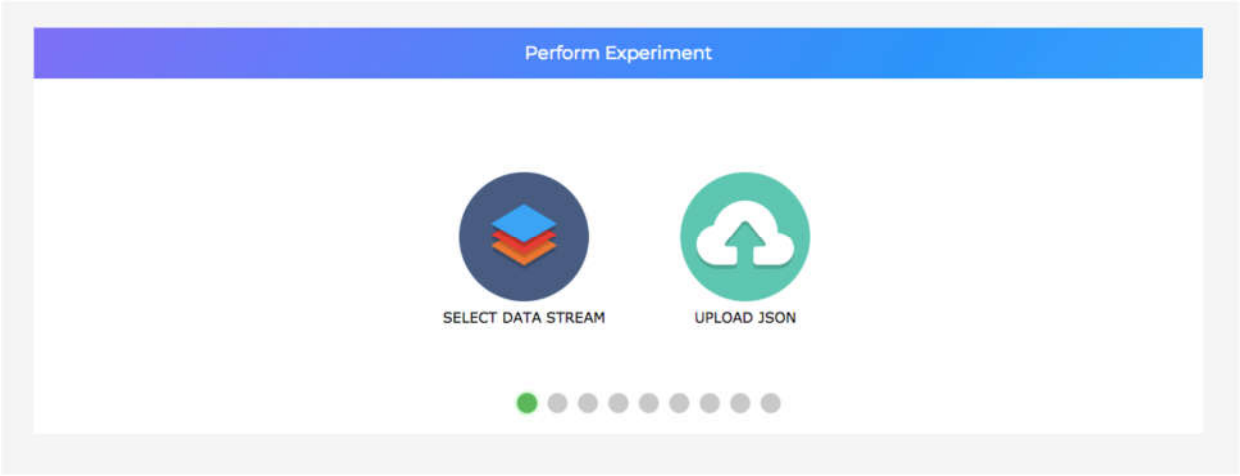

Рисунок 3.2 – Вікно вибору даних для проведення експерименту

На наступному кроці кроці користувач має можливість вибору типу великого потоку даних (рисунок 3.3).

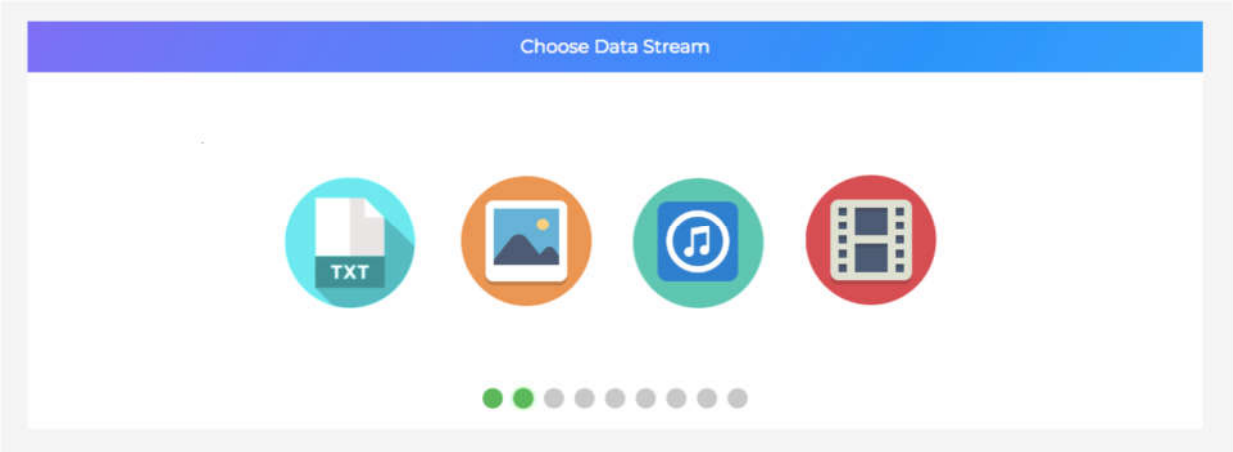

Рисунок 3.3 – Вікно вибору типу великих даних

Користувач має можливість вибрати джерело даних для отримання потоку даних (рисунок 3.4).

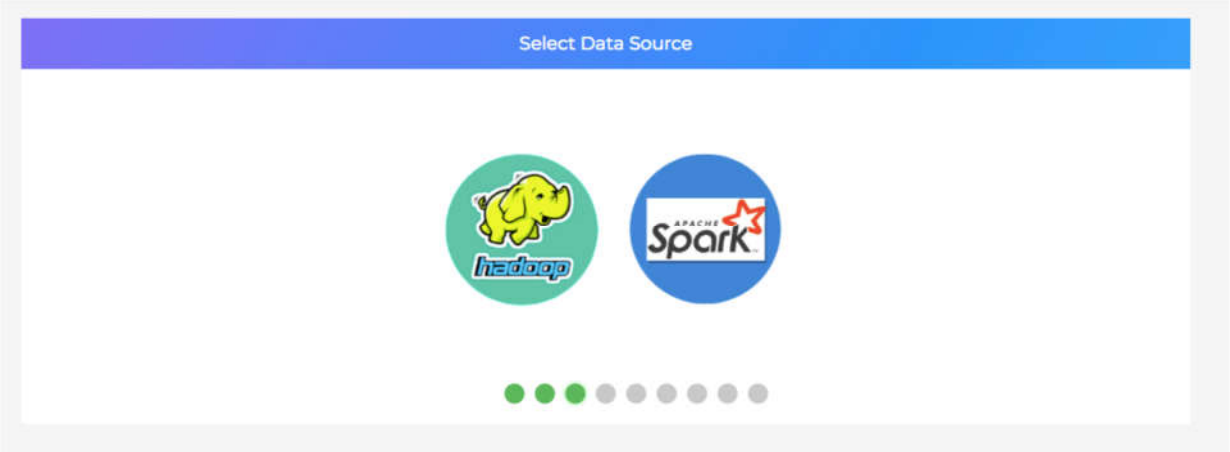

Рисунок 3.4 – Вікно вибору джерела даних

Вікно автентифікації користувача подано на рисунку 3.5.

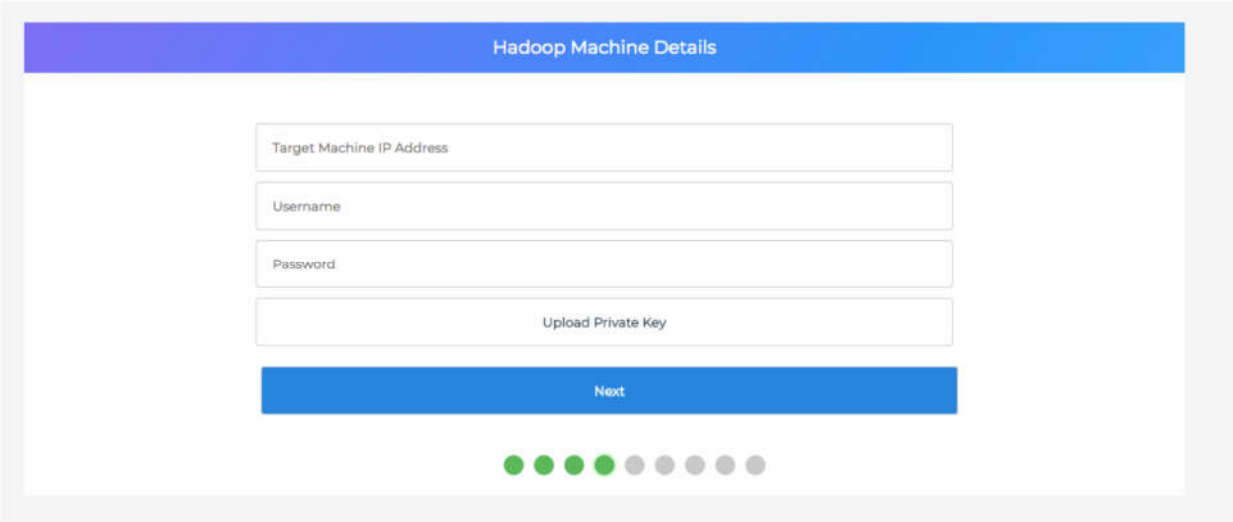

Рисунок 3.5 – Вікно автентифікації користувача

Вікно для відображення списку каталогів та файлів машини Hadoop представлено на рисунку 3.6.

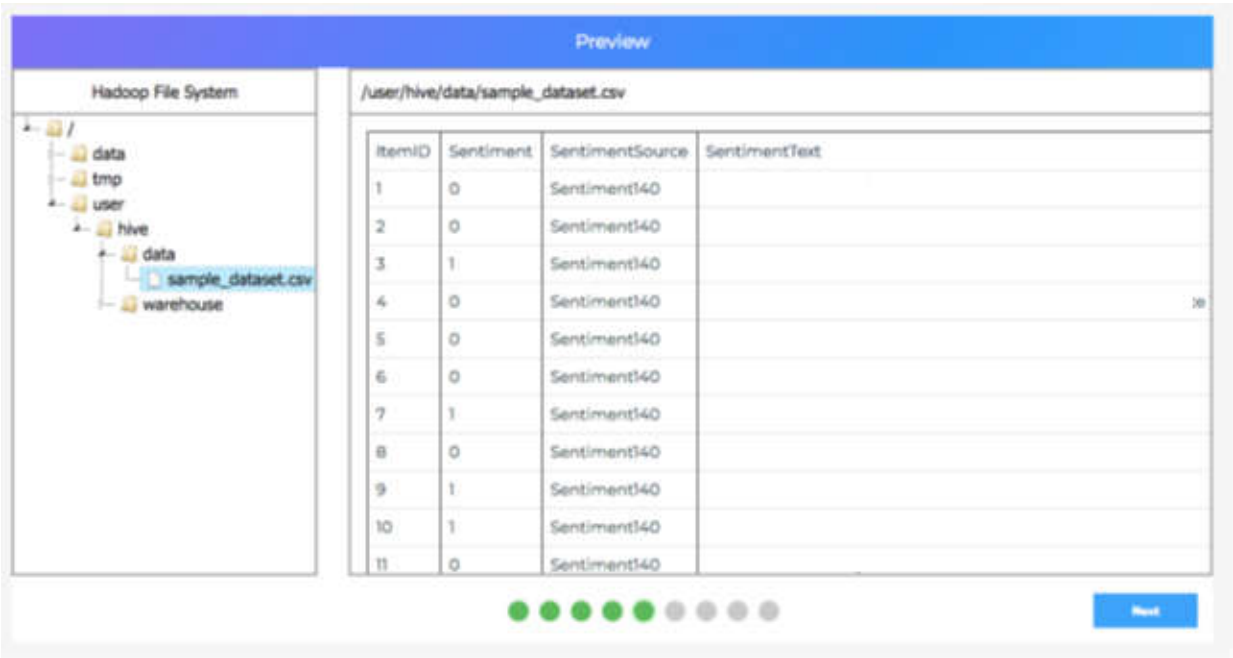

Рисунок 3.6 – Вікно відображення списку каталогів та файлів машини Hadoop

Вікно відображення операцій з текстовим файлом представлено на рисунку 3.7.

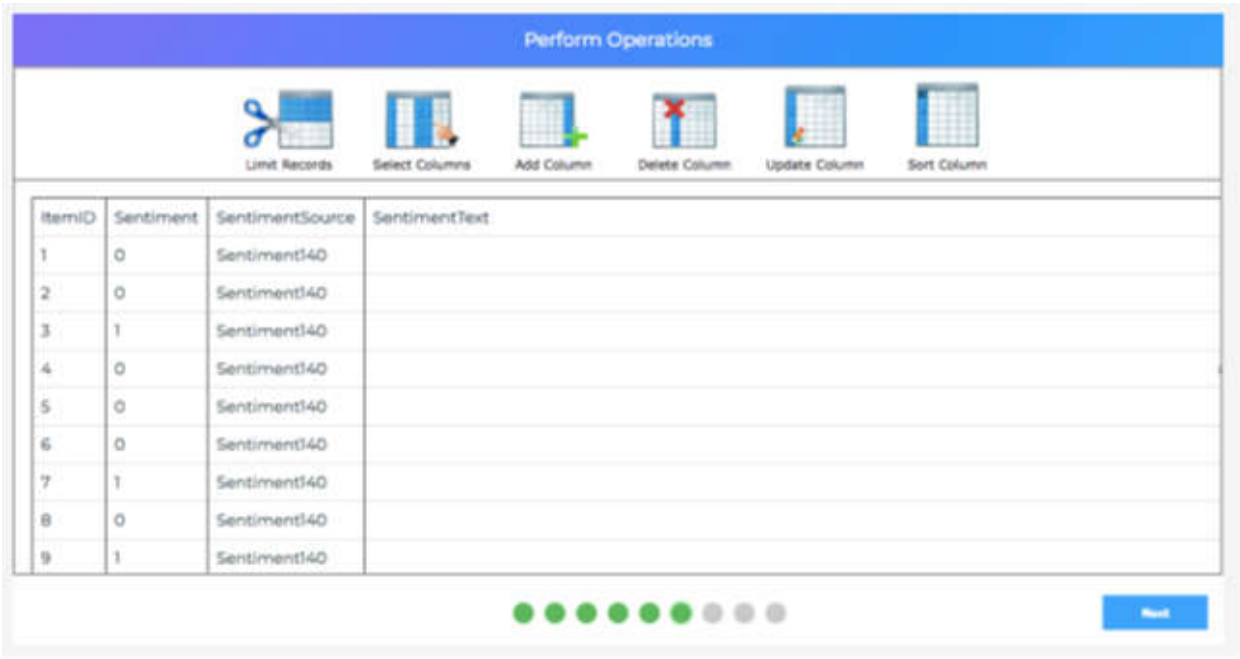

Рисунок 3.7 – Вікно відображення операцій з текстовим файлом

Вікно вибору задачі глибокого навчання подано на рисунку 3.8.

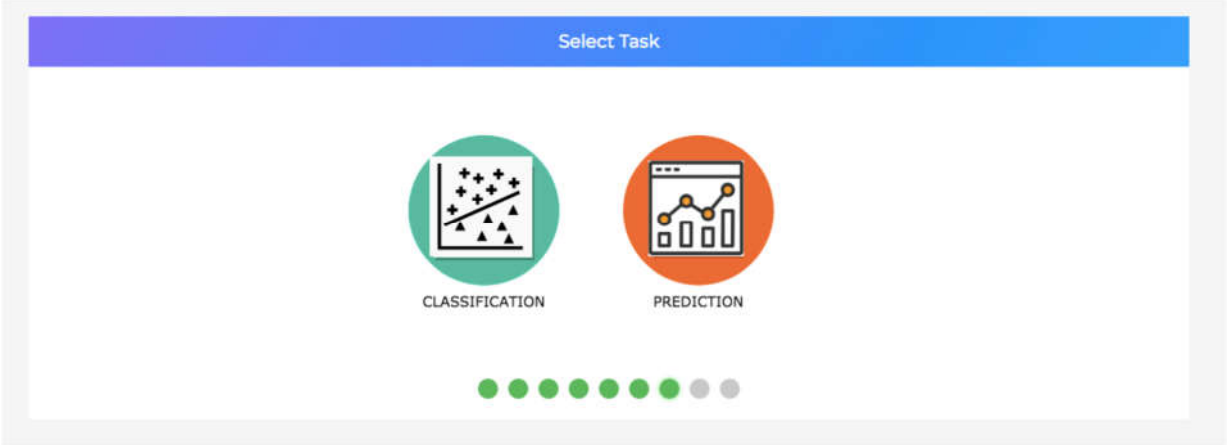

Рисунок 3.8 – Вікно вибору задачі глибокого навчання

Вікно вибору моделі глибокого навчання представлено на рисунку 3.9.

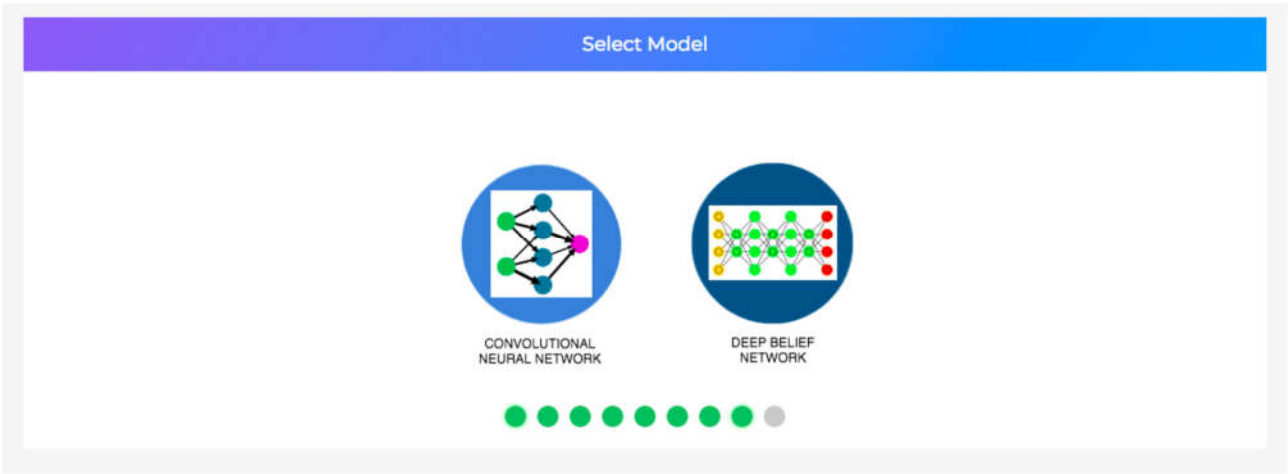

Рисунок 3.9 – Вікно вибору моделі глибокого навчання

Вікно налаштування гіперпараметрів моделі глибокого навчання зображено на рисунку 3.10.

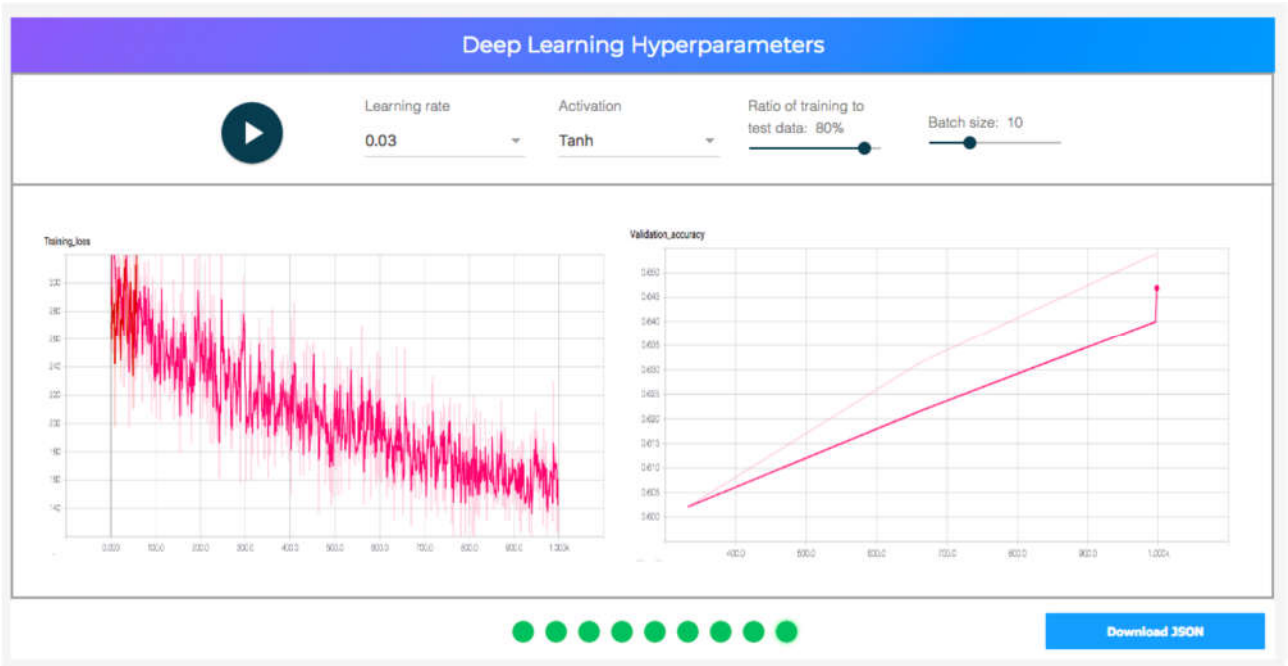

Рисунок 3.10 – Вікно налаштування гіперпараметрів моделі глибокого навчання

3.3 Оцінка ефективності запропонованого методу інтеграції великих наборів даних та моделей глибокого навчання

Розглянемо переваги використання запропонованого програмного забезпечення порівняно з ручним режимом.

Ручне виконання кроків можуть зайняти досить багато часу та спричинити помилки на кожному етапі роботи. Крім того, ручний режим займає багато часу, в залежності від задачі обробки та аналізу даних. Тут слід зазначити, що час на навчання глибокої нейронної мережі слід винести за межі порівняльно аналізу різних режимів, оскільки його оцінити дуже важко. Час навчання глибокої нейронної мережі дуже залежить від апаратного забезпечення, яке використовується та від складності нейронної мережі і алгоритмів її навчання.

На рисунках 3.10-3.11 показано порівняння ручного підходу та підходу на основі запропонованого ПЗ щодо швидкості роботи під час аналізу числових даних та зображень.

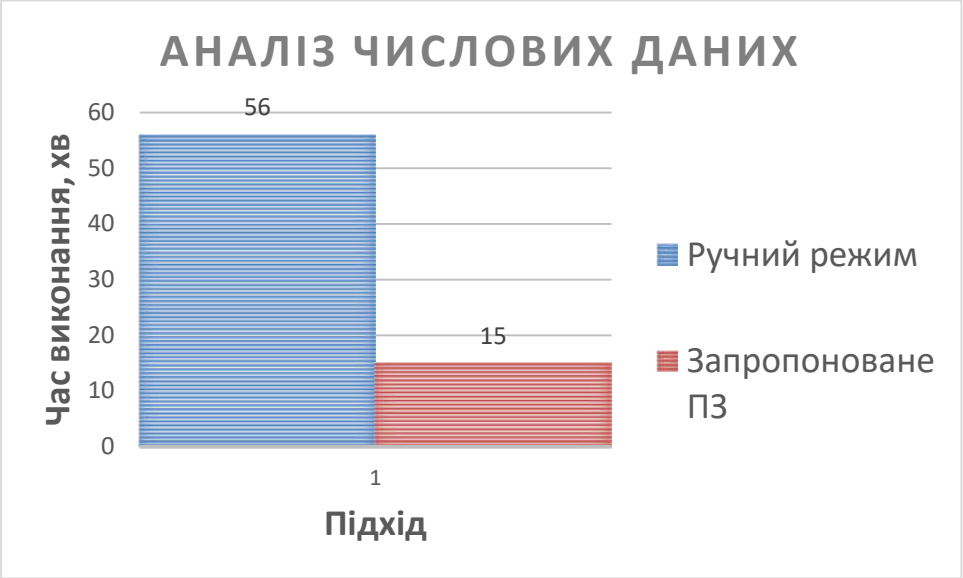

Рисунок 3.10 – Результати порівняння часу проведення експериментів ручним та запропонованим способами для задачі аналізу числових даних

Рисунок 3.11 – Результати порівняння часу проведення експериментів ручним та запропонованим способами для задачі аналізу зображень

Як видно з представлених рисунків швидкість проведення експериментів запропонованим способом, в порівнянні з ручним, вдалося збільшити в середньому в 3,6 рази.

Висновки до розділу 3

1. Розроблено архітектуру запропонованого програмного забезпечення інтеграції систем обробки великих даних та моделей глибокого навчання. Використовуючи запропонований підхід, користувачі зможуть трансформувати дані, що надходять з різних потоків великих даних, у формат, необхідний для навчання моделей глибокого навчання. Запропонований підхід забезпечить структуру, яка інтегрує різні потоки великих даних та моделі глибокого навчання, а також дозволить користувачам виконувати різні маніпуляції з потоками даних. Використовуючи цей фреймворк, користувачі зможуть швидко побудувати та

виконати різні експерименти з підмножиною набору даних, щоб отримати з нього значущі результати.

2. Розроблене програмне забезпечення надає користувачеві можливість вибору даних (пакетних даних/даних у реальному часі) з різних архітектур великих даних, попередньої обробки даних для перетворення даних у необхідний формат, а потім навчання моделей глибокого навчання з використанням попередньо оброблених даних. Швидкість проведення експериментів запропонованим методом, в порівнянні з ручним режимом, вдалося збільшити в середньому в 3,6 рази.

ВИСНОВКИ

В кваліфікаційній роботі запропоновано та реалізовано метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання.

Отримано наступні основні результати:

1. Проведено дослідження сучасного стану задачі обробки великих даних, яке показало недосконаліть і обмеженість існуючих систем розподіленої обробки великих даних.

2. Проведене дослідження підтвердило актуальність в розробленні нових підходів для інтеграції розподілених систем обробки великих даних та моделей глибокого навчання.

3. Проведено дослідження методів реалізації моделей глибокого навчання для обробки великих даних. Використання великих наборів даних на навчання архітектур нейронних мереж має великий вплив в багатьох областях, але потребує великих обчислювальних ресурсів. Сучасні досягнення мережевих архітектур, GPU та методів навчання дозволяють вирішити такі проблеми.

4. Вдосконалено метод інтеграції розподілених систем обробки великих даних з моделями глибокого навчання, що дозволить трансформувати дані, що надходять з різних потоків великих даних, у формат, необхідний для навчання глибоких нейронних мереж, а також дозволить виконувати різні маніпуляції з потоками даних.

5. Розроблено узагальнений алгоритм роботи методу в режимі обробки великих даних та узагальнений алгоритм роботи методу в режимі навчання нейромережевих моделей.

6. Розроблено архітектуру запропонованого програмного забезпечення інтеграції систем обробки великих даних та моделей глибокого навчання. Використовуючи запропонований підхід, користувачі зможуть трансформувати дані, що надходять з різних потоків великих даних, у

формат, необхідний для навчання моделей глибокого навчання. Запропонований підхід забезпечить структуру, яка інтегрує різні потоки великих даних та моделі глибокого навчання, а також дозволить користувачам виконувати різні маніпуляції з потоками даних. Використовуючи цей фреймворк, користувачі зможуть швидко побудувати та виконати різні експерименти з підмножиною набору даних, щоб отримати з нього значущі результати.

7. Розроблене програмне забезпечення надає користувачеві можливість вибору даних (пакетних даних/даних у реальному часі) з різних архітектур великих даних, попередньої обробки даних для перетворення даних у необхідний формат, а потім навчання моделей глибокого навчання з використанням попередньо оброблених даних. Швидкість проведення експериментів запропонованим методом, в порівнянні з ручним режимом, вдалося збільшити в середньому в 3,6 рази.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Al-Jarrah O. Y., Yoo P. D., Muhaidat S., Karagiannidis G. K., Taha K. Efficient machine learning for big data: A review. CoRR, 2015. abs/1503.05296.

2. Alsheikh M. A., Niyato D., Lin S., Tan H., Han Z. Mobile big data analytics using deep learning and apache spark. *IEEE Network*. 2016. Vol.30(3). P.22–29.

3. Apache Hadoop. URL: http://hadoop.apache.org/

4. Apache Mahout. URL: https://mahout.apache.org/

5. Apache Spark – Unified Analytics Engine for Big. URL: https://spark.apache.org.

6. Apache Storm URL: https://storm.apache.org.

7. Araki T., Nakamura Y. Future trend of deep learning frameworks- from the perspective of big data analytics and hpc. *In 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 2017. P. 696–703.

8. Bengio Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*. 2009. Vo1.2(1). P. 1-127.

9. Bengio Y., Lamblin P., Popovici D., Larochelle H. Greedy layer-wise training of deep networks. In B. Scholkopf, J. C. Platt, T. Hoffman (Eds.). *Advances in neural information processing systems*, MA: MIT Press, Cambridge. 2007. Vo1.11. P. 153-160.

10. Berthold M., Cebron N., Dill F., Gabriel T., Kötter T., Meinl T., Ohl P., Sieb C., Thiel K., Wiswedel B. Knime: The konstanz information miner. 2008. P. 319–326.

11. Big Data for the Enterprise URL: http://Big Datawithoracle-521307.pdf.

12. Dean J., Corrado G. S., Monga R., Chen K., Devin M., Le Q.V., Mao M.Z., Ranzato M., Senior A., Tucker P., Yang K., Ng A. Y. Large scale distributed deep networks. *In Proceedings of the 25th International Conference on Neural Information Processing Systems.* 2012. Vol. 1. P. 1223–1231.

13. Dwivedi S., Kasliwal P., Soni S. Comprehensive study of data analytics tools (rapidminer, weka, r tool, knime). 2016. P. 1–8.

14. Elgendy N., Elragal A. Big data analytics: a literature review paper. s.l. Springer, cham. 2014. P. 214-227.

15. Erhan D., Bengio Y., Courville A., Manzagol P.-A., Vincent P., Bengio S. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*. 2010. Vo1.11. P. 625-660.

16. Golovko V. A Learning Technique for Deep Belief Neural. *Communication in Computer and Information Science*. 2014. Vol. 440. P. 136–146.

17. Golovko V., Kroshchanka A., Rubanau U., Jankowski S. Learning Technique for Deep Belief Neural Networks. *Communication in Computer and Information Science*. 2014. Vol. 440. P. 136-146.

18. Golovko V. A New Technique for Restricted Boltzmann Machine Learning. *Proceedings of the 8th IEEE International Conference IDAACS.* 2015. P. 182–186.

19. Gupta A., Thakur H. K., Shrivastava R., Kumar P., Nag S. A big data analysis framework using apache spark and deep learning. *In 2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2017. P. 9–16.

20. Hbibi L., Barka H. Big data: Framework and issues. Proceedings of the 2016 International Conference on Electrical and Information Technologies. 2016. P. 485–490.

21. Hinton G. E. A practical guide to training restricted Boltzmann machines. Machine Learning Group, University of Toronto. 2010 (Tech. Rep. 2010-000).

22. Hinton G. E., Osindero S., Teh Y. A fast learning algorithm for deep belief nets. *Neural Computation*. 2006. Vo1. 18. P. 1527-1554.

23. Hinton G. Training products of experts by minimizing contrastive divergence. *Neural Computation*. 2002. Vo1. 14. P. 1771-1800.

24. Hinton G., Salakhutdinov R. Reducing the dimensionality of data with

neural networks. *Science*. 2006. Vo1. 313 (5786). P. 504-507.

25. HIVE. URL: https://hive.apache.org/

26. Khumoyun A., Cui Y., Hanku L. Spark based distributed deep learning framework for big data applications. *2016 International Conference on Information Science and Communications Technologies (ICISCT)*. 2016. P. 1–5.

27. KNIME. URL: https://www.knime.com/

28. Koliopoulos A. K., Yiapanis P., Tekiner F., Nenadic G., Keane J. A parallel distributed weka framework for big data mining using spark. 2015. P. 9–16.

29. Kotsiantis S.B., Kanellopoulos D., Pintelas P.E. Data preprocessing for supervised leaning. *International Journal of Computer Science*. 2006. Vol. 1(2). P.111–117.

30. LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature*. 2015. Vol. 521 (7553). P. 436–444.

31. Lee S., Kim J. K., Zheng X., Ho Q., Gibson G. A., Xing E. P. On model parallelization and scheduling strategies for distributed machine learning. *Advances in Neural Information Processing Systems (NIPS 2014)*. 2014. P. 2834–2842.

32. Liu W., Z. Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*. 2017. Vol. 234. P.11–26.

33. Lynch C. Big data: science in the petabyte era. *Nature*. 2008. Vol. 455. P. 1-50.

34. Mayer-Schonberger V., Padova Y. Regime Change: Enabling Big Data through Europe's New Data Protection Regulation. *Colum. Sci. & Tech. L. Rev. Journal*. 2015. P. 315.

35. Najafabadi M. M., Khoshgoftaar T. M., Villanustre F., Holt J. Large-scale distributed l-bfgs. *Journal of Big Data*. 2017. Vol. 4(1). P. 22-38.

36. Prekopcsák Z., Makrai G., Henk T., Gáspár-Papanek C. Radoop: Analyzing Big Data with RapidMiner and Hadoop. 2011.

37. Radoop. URL: https://rapidminer.com/products/radoop/

38. Raina R., Madhavan A., Ng A. Y. Large-scale deep unsupervised learning using graphics processors. *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009. P. 873–880.

39. RapidMiner. URL: https://rapidminer.com/

40. Shi S., Chu X. Performance modeling and evaluation of distributed deep learning frameworks on gpus. *CoRR*. 2017. abs/1711.05979.

41. Thiel K. The knime text processing plugin. 2009.

42. Tien J.M. Big data: Unleashing information. *Journal of Systems Science and Systems Engineering*. 2013. Vol. 22(2). P. 127-151.

43. Wang H., Xu Z., Fujita H., Liu S. Towards felicitous decision making: An overview on challenges and trends of Big Data. *Information Sciences journal*. 2016. Vol. 367. P. 747-765.

44. WEKA. URL: https://www.cs.waikato.ac.nz/ml/weka/

45. White C. Using big data for smarter decision making IBM. Yorktown Heights, New York. 2011.

46. Witten I. H., Frank E., Trigg L., Hall M., Holmes G., Cunningham S. J. Weka: Practical machine learning tools and techniques with java implementations. 1999.

47. Головко В.А. От многослойных персептронов к нейронным сетям глубокого доверия: парадигмы обучения и применение. *XVII Всероссийская научно-техническая конференция «Нейроинформатика-2015»: Лекции по Нейроинформатике*. 2015. С. 47-84.

48. Джулли А., Пал С. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow. ДМК-Пресс. 2017. 294 с.

49. Жерон О. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. Вильямс. 2018. 688 с.

50. Коэльё Л. П., Ричерт В. Построение систем машинного обучения на

языке Python. Перевод с английского. М.: ДМК Пресс. 2015. 303 с.

51. Майер-Шенбергер В., Кукьер К. Большие данные. Революция, которая изменит то, как мы живем, работаем и мыслим. пер. с англ. И. Гайдюк. М.: Манн, Ивановы Фербер, 2014. 240 с.

52. Маккинли У. Python и анализ данных. Перевод с английского. М.: ДМК Пресс. 2015. 482 с.

53. Онищенко І.М. Удосконалення методів обробки та зберігання даних за допомогою інструментів «Big Data» та Map Reduce. *Економіко-математичне моделювання соціально-економічних систем*. Збірник наукових праць. Київ. 2017. № 22. С. 159-178.

54. Риза С., Лезерсон У., Оуэн Ш., Уиллс Д. Spark для профессионалов: современные паттерны обработки больших данных. Питер. 2017. 272 с.

55. Уайт Т. Hadoop. Подробное руководство. СПб.: Питер. 2013. 672 с.

56. Івахів В.В., Ляпандра І.А., Білоус В.С. Актуальність обробки та аналізу великих даних. Збірник тез доповідей міжнародної наукової інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення». Вип. 63. 11 листопада 2021 р. С.12-14.

57. Ляпандра І.А., Івахів В.В., Білоус В.С. Методи та засоби обробки великих даних // Збірник тез доповідей X міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій». Тернопіль, ТНТУ, 2021 р. Т.0. С.00-00.

58. Загальні рекомендації з підготовки, оформлення, захисту та оцінювання випускних кваліфікаційних робіт здобувачів вищої освіти першого «бакалаврського» і другого «магістерського» рівнів / За ред. доц. М.І. Шинкарика. Тернопіль: ТНЕУ, 2018. 67 с.

59. Комар М.П., Саченко А.О., Васильків Н.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2021. 32 с.