Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи аналізу ефективності навчання модифікованих штучних нейронних мереж

(тема)

Виконав:

студент II курсу, групи СПзм-20-1

Синяокий А.О.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Козлов Ю.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Синяокому Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи аналізу ефективності навчання модифікованих штучних нейронних мереж

затверджена наказом по університету від " 25 " березня 2022 р. № 33 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи
паралельні алгоритми
карти Кохонена
штучні нейронні мережі
кластеризація

4. Перелік питань, що потрібно опрацювати у роботі
Паралельні алгоритми аналізу даних
Методи побудови паралельних алгоритмів з використанням ШНМ
Реалізація модифікованого методу побудови паралельних алгоритмів аналізу даних

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання.Аналіз предметної області	25.03.2022–13.04.2022	
2	Аналіз існуючих моделей та методів	14.04.2022–26.04.2022	
3	Розробка методу	27.04.2022–29.04.2022	
4	Проведення моделювання	30.04.2022–04.05.2022	
5	Отримання та аналіз результатів	05.05.2022–07.05.2022	
6	Оформлення пояснювальної записки	08.05.2022–13.05.2022	

Дата видачі завдання 25 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Козлов Ю.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 77 с., 26 рис., 1 дод., 21 джерело.

ШТУЧНА НЕЙРОННА МЕРЕЖА, ЕФЕКТИВНІСТЬ, КАРТА КОХОНЕНА, МОДИФІКАЦІЯ, ПАРАЛЕЛЬНИЙ АЛГОРИТМ.

Метою кваліфікаційної роботи є аналіз методів ефективності навчання модифікованих штучних нейронних мереж.

У ході виконання кваліфікаційної роботи проведено аналіз методів ефективності навчання модифікованих штучних нейронних мереж типу карт Кохонена. Виконано аналіз існуючих засобів кластеризації, в тому числі розподілених даних, який показує те, що основні вимоги для розподілених систем моніторингу відповідають алгоритмам кластеризації, що використовують нейронні мережі Кохонена. Розроблена формальна модель декомпозиції алгоритмів кластеризації, використовують нейронні мережі Кохонена для горизонтально і вертикально розподілених даних. Розроблено метод об'єднання проміжних результатів отриманих при аналізі розподілених даних з урахуванням типу їх розподілу для алгоритмів кластеризації, що використовують нейронні мережі Кохонена. Виконана програмна реалізація алгоритму кластеризації, що використовує модифіковані нейронні мережі Кохонена для обробки розподілених даних з урахуванням методу об'єднання отриманих результатів.

ABSTRACT

Master’s thesis: 77 pages, 26 figures, 1 appendices, 21 sources.

ARTIFICIAL NEURAL NETWORK, EFFICIENCY, KOHONEN MAP, MODIFICATION, PARALLEL ALGORITHM.

The major goal of this thesis is to analyze the effectiveness of training methods of modified artificial neural networks.

In order to analysis of the effectiveness of learning methods of modified artificial neural networks of the Kohonen map type was carried out. An analysis of existing clustering tools, including distributed data, is performed, which shows that the main requirements for distributed monitoring systems correspond to clustering algorithms using Kohonen neural networks. A formal decomposition model of clustering algorithms is developed, using Kohonen neural networks for horizontally and vertically distributed data. A method of combining intermediate results obtained during the analysis of distributed data, taking into account the type of their distribution for clustering algorithms using Kohonen neural networks, has been developed. The software implementation of the clustering algorithm, which uses modified Kohonen neural networks for processing distributed data, taking into account the method of combining the obtained results, was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІАД – інтелектуальний аналіз даних

КАКК – класичний апарат карт Кохонена

МН – машинне навчання

ПЗ – програмні засоби

РОД – розподілена обробка даних

ВСТУП

За останні роки у людства накопичені значні обсяги оцифрованих даних, які містять корисні знання. Вони розміщуються в сховищах даних або на безлічі територіально-розподілених вузлів, об'єднаних як локальними, так і глобальними засобами передачі даних, що утворюють розподілені обчислювальні середовища. З розвитком технології Інтернету речей (Internet of Things), джерелами інформації стають не тільки люди, але і пристрої (сенсори, смартфони, відеокамери і т.п.) підключаються до мережі. Вони формують потоки різнорідних даних, збільшуючи кількість розподілених джерел інформації. Лавиноподібне зростання джерел і обсягів інформації, її різнорідність і розподілений характер зберігання привели до необхідності перегляду не тільки технологій збору і зберігання даних, але також і технологій аналізу данних.

Аналіз даних дозволяє формувати нові знання, що містять уявлення про характер об'єктів або явищ. До типових завдань аналізу даних відносяться: класифікація, кластеризація, знаходження асоціацій, виявлення аномалій, і ін. Для вирішення таких завдань розроблено велику кількість алгоритмів аналізу даних. Більшість з відомих алгоритмів аналізу застосовуються до даних, що зберігаються централізовано. Використання таких алгоритмів у випадках, коли дані розміщені на розподілених джерелах, передбачає їх наявність водному узлехраненія даних, що призводить до наступних проблем:

- витрачається час на передачу інформації, що може бути критично при її обробці, наприклад, в режимі реального часу;

- зростає мережевий трафік, що знижує можливості використання каналів зв'язку з низькою пропускною здатністю (супутникових каналів зв'язку, бездротових каналів і т.п.);

- потрібна передача інформації, в тому числі конфіденційної, по відкритих каналах передачі даних;

- підвищується цінність зібраної в одному місці інформації, що вимагає підвищених заходів забезпечення її безпеки та надійності.

Виключити вказані недоліки можна за рахунок аналізу безпосередньо на джерелі інформації. Подібний підхід передбачає також активно розвивається концепція «туманних» обчислень (Fog Computing) [1]. Однак для реалізації даної концепції в області аналізу даних необхідно створення нових методів і засобів побудови паралельних алгоритмів, що дозволяють виконувати аналіз великих обсягів розподілених даних без їх попереднього збору в єдиному сховищі. Крім того, нові методи і засоби повинні враховувати тіпраспределенія даних. Дані можуть бути розподілені горизонтально, коли на різних джерелах зберігається інформація про різні об'єкти та явища, або вертикально, коли на різних джерелах зберігається інформація про різні характеристики одних і тих же об'єктів.

В області побудови паралельних алгоритмів аналізу даних можна виділити два основні підходи. Перший підхід спрямований на розпаралелювання відомих і перевірених на практиці послідовних алгоритмів, в тому числі, і алгоритмів аналізу даних.

Основним недоліком такого підходу є складність розпаралелювання, пов'язана з можливою відсутністю внутрішнього паралелізму в таких алгоритмах. Це призводить до високої трудомісткості розпаралелювання алгоритмів для різних умов виконання (наприклад, для роботи із загальною або розподіленою пам'яттю, для аналізу горизонтально або вертикально розподілених даних і т.п.). Альтернативним підходом є побудова нових паралельних алгоритмів аналізу даних, оптимізованих для виконання в заданих умовах.

Недоліком такого підходу є необхідність перевірки коректності нових алгоритмів на наборах даних з різними характеристиками. Крім того, при зміні умов виконання потрібна розробка нових алгоритмів, що веде до необхідності нових исследованій.На рівні розподілених компонент дослідження спрямовані, в тому числі, на створення нових методів і засобів,

що оптимізують виконання паралельних алгоритмів в різних розподілених середовищах і підвищують можливості їх масштабування.

Існуючі моделі і методи побудови паралельних алгоритмів аналізу даних не враховують тіпраспределенія даних і припускають їх застосування до єдиного джерела інформації. Вони дозволяють знизити час виконання алгоритму аналізу, але в разі розподіленого зберігання даних необхідність їх збору в єдиному сховищі вимагає додаткових витрат часу, збільшує мережевий трафік і підвищує вимоги до безпеки. Бачиться актуальним дослідити існуючі моделі і методи паралельного побудови алгоритмів аналізу даних для виконання в розподіленої середовищі з можливістю розміщення частини функцій аналізу на джерелах інформації. При цьому враховується тип розподілу даних, що підвищує продуктивність, знижує мережевий трафік і не вимагає підвищених заходів безпеки.

Метою кваліфікаційної роботи є аналіз методів ефективності навчання модифікованих штучних нейронних мереж.

Об’єкт дослідження: модифіковані штучні нейронні мережі.

Завдання:

- аналіз існуючих моделей, методів та засобів інтелектуального аналізу даних з використанням штучних нейронних мереж;
- аналіз існуючих засобів кластеризації, в тому числі розподілених даних;
- розробка методу виконання кластеризації на основі модифікованих нейронних мереж Кохонена на розподілених джерелах, даних з урахуванням типу розподілу даних;
- програмна реалізація алгоритмів кластеризації.

1 ПАРАЛЕЛЬНІ АЛГОРИТМИ АНАЛІЗУ ДАНИХ

Теперішній час характеризується широким застосуванням цифрових технологій і загальної інформатизацією суспільства. В результаті у людства накопичені величезні обсяги даних. Однак без їх аналізу та вилучення нових корисних знань онине представляють особойценності. За кілька останніх десятиліть було розроблено велику кількість методів і алгоритмів, як послідовних, так і паралельних, успішно вирішують різні завдання аналізу даних: класифікації, кластеризації, виявлення аномалій і т.п.

Вони успішно застосовуються до даних зберігаються в базах даних, сховищ даних і т.п. У той же час, аналіз потрібно не тільки по відношенню до даних, попередньо зібраним і зберігаються централізовано, а й до розподілених даних, атакож до даних надходять від різних пристроїв: відео- та фотокамер, датчиків, сенсорів і ін. З розвитком технології Інтернету речей, збільшується число пристроїв, що об'єднуються в мережі. За інформацією видання Gartner1к 2022 році до Інтернету буде підключено близько 35 мільярдів устройств. Внових условіяхтребуется не тільки підвищення швидкодії алгоритмів аналізу даних, за рахунок їх розпаралелювання, але і їх виконання в розподілених середовищах з безліччю джерел інформації. Дослідження в області паралельних і розподілених обчислень ведуться давно на різних рівнях. У цьому розділі розглянемо аналіз існуючих методів і засобів побудови паралельних алгоритмів, в тому числі алгоритмів аналізу даних.

1.1 Методи аналізу даних

Під аналізом даних (інформації) розуміють сукупність дій здійснюваних дослідником в процесі вивчення отриманих тим чи іншим чином даних з метою формування певних уявлень про характер явища, що

описується цими даними. Такі уявлення необхідні для класифікації явища, його асоціювання і т.п.

Алгорітми аналізу даних обробляють масиви інформації з метою отримання нових знаній. К найбільш раннім методам аналізу даних можна віднести методи математичної статистики. Вони застосовувалися до числових даних для розрахунків різних параметрів: математичного очікування, дисперсії, трендів, апроксимації тощо.

Розвитком цих методів є методи інтелектуального аналізу: методи машинного навчання (machine learning), data mining, глибинний аналіз (deep learning) і т.п. Інтеллектуальний аналіз даних (ІАД) - це процес виявлення в «сирих» даних раніше невідомих нетривіальних практично корисних і доступних інтерпретації знань, необхідних для прийняття рішень в різних сферах людської деятельності. Формальне представлення витягнутих знань будемо називати моделлю знань. Витягнуті моделі знань дозволяють виконувати різні аналітичні функції, основними з них є:

- класифікація - визначення класу об'єкта по його характеристиках (безліч класів, до яких може бути віднесений об'єкт, заздалегідь відомо);
- регресія - визначення за відомими характеристиками об'єкта значення деякого його параметра (на відміну від задачі класифікації значенням параметра є не кінцеве безліч класів, а множина дійсних чисел);
- пошук частих наборів - знаходження частих залежностей (асоціацій) між об'єктами або подіями;
- кластеризация - пошук незалежних груп (кластерів) і їх характеристик у всьому безлічі аналізованих даних;
- аналіз часових рядів - обчислення статистичних та інших характеристик даних змінюються в часі;
- виявлення аномалій - пошук викидів і аномальних значень в даних.

1.1.1 Модель знань

Знання, які добуваються алгоритмами аналізу даних, представляютсядля їх використання в системах підтримки прийняття рішень. Існують різні способи формального представлення знань:

- продукційні моделі;
- семантичні мережі;
- фрейми;
- логічні моделі.

Продукційна модель (модель, заснована на правилах) дозволяє представити знання у вигляді пропозицій, званих продукціям, виду: «якщо (умова), то (дія)».

Під умовою (антецедентом) розуміється деякий пропозицію-зразок. Під «дією» (Консеквентні) понімаетсяоперація, виполняемаяпрі успішному результаті умови. Перевагою продукционной моделі є наочність, висока модульність, легкість внесення доповнень і змін і простотою механізм логічного висновку. Недоліком є накопичення досить великої кількості (близько декількох сотень) продукцій, якіможуть суперечити один одному. Зростання суперечливості продукционной моделі може бути обмежений шляхом запровадження механізмів ісключенійі повернень.

Механізм винятків означає, що вводяться спеціальні правила-виключення. Їх відрізняє велика конкретність в порівнянні з узагальненими правилами. При наявності виключення основне правило не застосовується. Механізм повернень означає, що логічний висновок може тривати навіть в тому випадку, якщо на якомусь етапі висновок привів до протиріччя: просто необхідно відмовитися від одного з прийнятих ранееутвержденій і здійснити повернення до попереднього стану.

Семантична мережа - це орієнтований граф, вершини якого відображають деякі поняття предметної області, а дуги - відносини між ними. Таким чином, семантична мережа відображає семантику предметної області

у вигляді понять і відносин. Ідея систематизації на основі будь-яких семантичних відносин не раз виникала в ранні періоди розвитку науки. Прабатьками сучасних семантичних мереж можна вважати екзистенційні графи (existentialgraph), запропоновані Чарльзом Пірсом (Charles Sanders Peirce) в 1909 р.

Комп'ютерні семантичні мережі були детально розроблені Річардом Річенс в 1956 році в рамках проекту Кембриджського центру вивчення мови з машинного перекладу. Кількість типів відносин в семантичної мережі визначається її творцем виходячи з конкретних цілей. У реальному світі їх число прагне до нескінченності.

Фрейм це абстрактний образ для представлення некоторогостереотіпа інформації. Подання знань у вигляді фреймів, організованих в деревоподібну структуру було запропоновано М.Минская в 1979. Розрізняють фрейми-зразки (прототипи), що зберігаються в базі знань, і фрейми-екземпляри, які створюються для відображення реальних фактичних ситуацій на основі даних, що надходять. Модель фрейма є досить універсальной, оскільки дозволяє відобразити все різноманіття знань про світ через:

- фрейми-структури, що використовуються для позначення об'єктів і понять (позика, застава, вексель);
- фрейми-ролі (менеджер, касир, клієнт);
- фрейми-сценарії (банкрутство , збори акціонерів, святкування іменин);
- фрейми-ситуації (тривога, аварія, робочий режим пристрою) і ін.

Логічні моделі ґрунтуються на класичному численні предикатів І-го порядку, коли предметна область або завдання описуються у вигляді набору аксіом. Найчастіше ці логічні моделі будуються за допомогою декларативних мов логічного програмування, найбільш відомим представником яких, є мова Пролог (Prolog). Вони зручні для представлення логічних взаємозв'язків між фактаміі хорошоформалізовани. Перераховані способи представлення знань призначенідля опису знань про деяку предметну область і призначені для

досить широкого застосування. На відміну від них алгоритми аналізу витягають нові знання, що відображають закономірності в аналізованих даних. Для їх уявлення використовують різні моделі:

- класифікаційні правила (продукційні моделі);
- дерева рішень;
- нейронні мережі;
- математичні функції і т.п.

В області ІАД для опису різних моделі знань використовується стандарт PMML. Він описує кожну модель знань на мові XML. Основний конструкцією мови XML є елементи, які можуть містити в собі інші вкладені конструкції і тим самим формувати ієрархічну структуру у вигляді дерева. XML елементи можуть мати атрибути (щоб не плутати їх з атрибутами набору даних, будемо називати їх властивостями). Основним недоліком даного стандарту є не універсальність (для кожної виду знань вводиться своя модель). Крім того, він не передбачає паралельне побудова таких моделей.

1.1.2 Методи аналізу даних

Існує велика безліч алгоритмів, що будують різні моделі знань. Так, різні алгоритми класифікації будують різні моделізнаній (класифікатори). Виділяють такі типи класифікаторів (рисунок 1.1):

- класифікаційні правила являють собою набір правил виду «якщо-то», в яких в умовній частині записуються незалежні атрибути, а в заключній - значення цільових атрибутів, і будуються алгоритмами: 1R, DataSqueezer, сімейства RULES і ін;

- байєсова мережа містить для кожного незалежного атрибута і його значення вірогідності відношення вектора до того чи іншого класу і будується алгоритмом Naive Bayes і його модифікаціями: selective Naive Bayes, semiNaive Bayes, one-dependence Bayesian classifiers, K-dependence Bayesian classifiers, Bayesian network-augmented Naive Bayes, unrestricted

Bayesian classifiers, і Bayesian multinets;

- дерева рішень є дерева, в вузлах яких вказуються незалежні атрибути, а в листі - значення залежних атрибутів; будується алгоритмами: RPART, ID3 (Iterative Dichotomiser 3), C4.5, CART (Classification and Regression Trees) , CHAID (chi-squared automatic interaction detector), MARS , FACT , CRUISE , GUIDE , QUEST, CTree та ін .;

- найближчі сусіди є векторами найближчими до заданого вектора і будуються алгоритмами: KNN алгоритм [83], Wavelet Based K-Nearest Neighbor Partial Distance Search (WKPDS) алгоритм [84], Equal-Average Nearest Neighbor Search (ENNS) алгоритм, Equal-Average Equal-Norm Nearest Neighbor code word Search (EENNS) алгоритм, Equal-Average Equal-Variance Equal-Norm Nearest Neighbor Search (EEENNS) алгоритм;

- математична функція (регресія) являє собою функцію апроксимації, що будується алгоритмами: найменших квадратів, SVM і його модифікації GSVM (granular support vector machines)

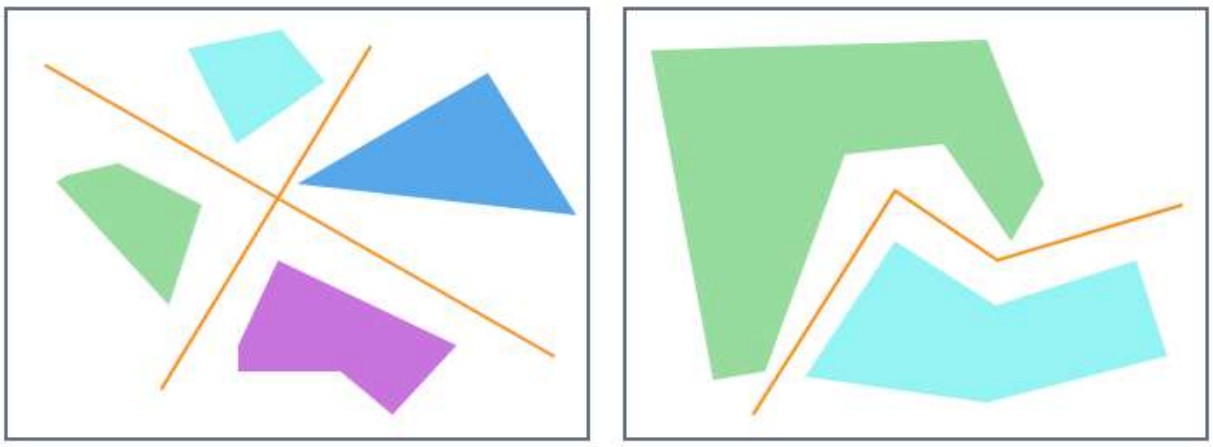

Рисунок 1.1 – Алгоритми класифікації

Алгоритми кластеризації будують моделі, які описують кластери різними способами (рисунок 1.2).

Розрізняють такі кластерні моделі:

- ієрархічні, що формують ієрархії кластерів (часто подаються у

вигляді дендрограмм) і будуються: о агломеративного алгоритмами (будують дендрограму від низу до верху): Ward's, UPGMA, SLINK, CURE (Clustering Using Representatives), дівізімнимі алгоритми(будують Дендрограмма зверху вниз) SVD (Singular Value Decomposition), DIANA;

- центроїдні, що визначають кластери за допомогою його центру кластера і споруджувані алгоритмами: Lloyd's, SNOB, MCLUST, k- means, BANG ;

- засновані на розподілі, визначальним для кожного вектора ймовірності, з якими він відноситься до кластерів: ЕМ алгоритм і його різновиди;

- щілісні, що описують кластери як області високої щільності (згустки); при цьому вектора, але вони не попадають в ці області, розглядаються як шум або кордону кластерів.

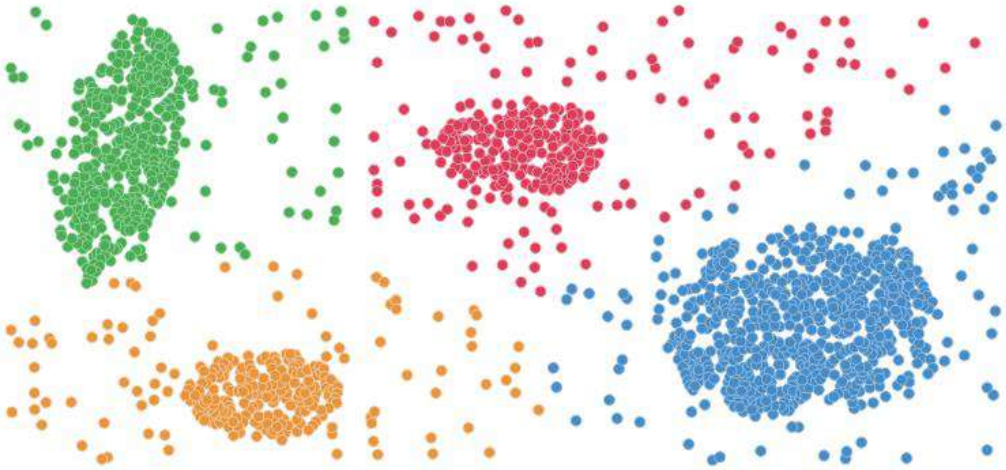

Рисунок 1.2 – Кластеризація даних

Алгоритми побудови асоціативних правил можна розділити на наступні категорії (рисунок 1.3):

- алгоритм Apriori і його модифікації: Partition, MSPS, LAPIN-SPAM та ін. алгоритми, що будують граф (в тому числі дерево) для визначення частих наборів: FP-Growth, ECLAT, FIN, PrePost, PPV та ін .;

- алгоритми, що використовують хешування: DHP, CARMA;

- алгоритми аналізу послідовності подій (sequence analyses), наприклад, алгоритми PROWL та ін.

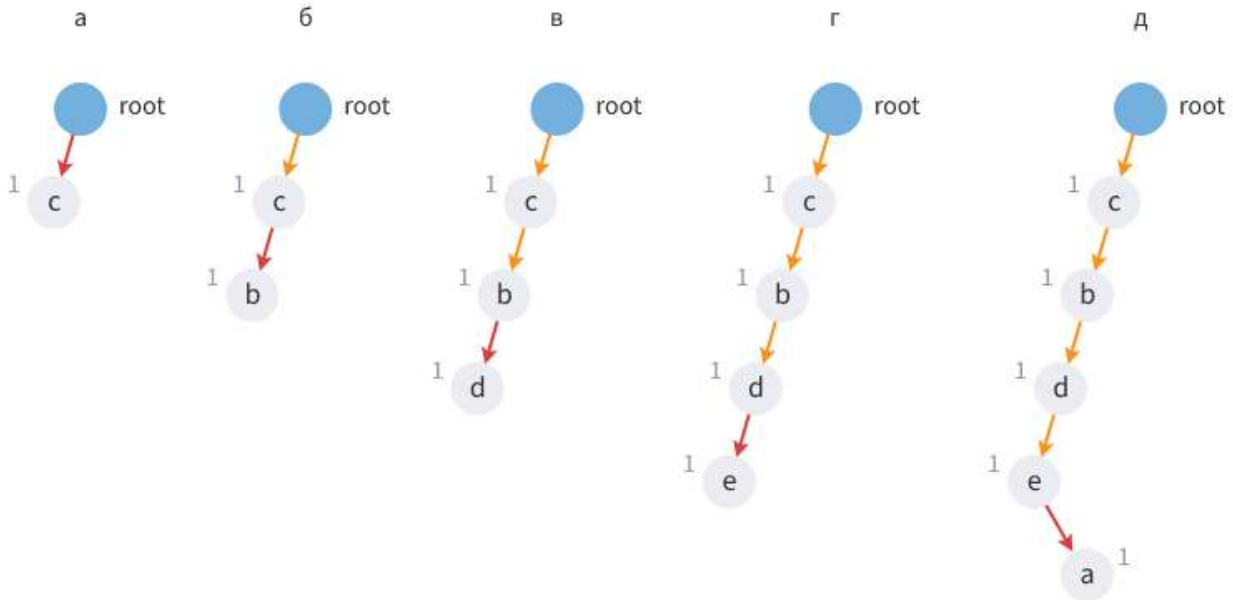

Рисунок 1.3 – Алгоритми пошуку асоціативних правил

Алгоритми аналізу часових рядів діляться на три частини (рисунок 1.4):

- уявлення тимчасового ряду: о засновані на представленні моделі: ARMA, time series bitmaps;

- ті, що не адаптуються до даних: DFT, wavelet functions related topic і PAA;

Ті, що адаптуються до даних: DFT / PAA і індексований PLA ;

- симулюють (апроксимуючі вимірювання): subsequence matching і full sequence matching;

- уявлення індексуючі: SAMs (SpatialAccess Methods) і TS-Tree.

Алгоритми виявлення аномалій шукають шаблони в даних і потім при появі даних, що не покриваються такими шаблонами, вважають їх викидами і аномаліями.

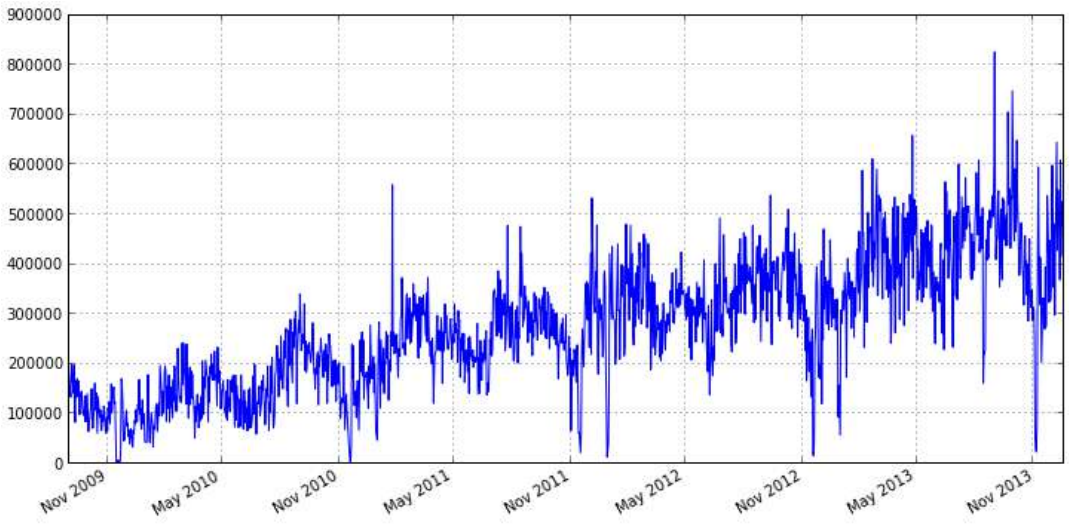

Рисунок 1.4 – Алгоритми аналізу часових рядів

Прикладом такого алгоритму є fuzzy rough sets to identify outliers. Незважаючи на велику різноманітність алгоритмів аналізу даних можна виділити наступні їх загальні властивості:

- незмінність аналізованих даних;
- формування моделі знань, розмір якої значно менше обсягу аналізованих даних;
- необхідність одного або декількох проходів як по об'єктах, що описуються даними, так і за їх характеристиками;
- витяг з даних закономірностей складними функціями, які обробляють кілька довільних об'єктів з набору даних (що не володіють властивістю облікового гомоморфизма).

1.2 Розподілена обробка даних

Зростання обсягів інформації, що зберігається привів до появи поняття Великих даних, які часто визначаються трьома V: Volume, Variety і Velocity):

- великі обсяги інформації (data volume);
- різні формати (data variety);
- генеруються з високою частотою (data velocity). Аналіз таких даних є

ресурсномісткої завданням. У зв'язку з цим для обробки і аналізу Великих даних використовують засоби розподілених обчислень, що реалізують різні моделі.

Найбільш популярною в даний час є програмна модель розподілених обчислень MapReduce. Вона розділяє обробку даних на дві функції, які реалізуються розробником: map і reduce:

- map - функція приймає два аргументи: ключ k1 і значення v1 і повертає набір проміжних пар ключ / значення [(k2, v2)]: map :: k1 ^ v1 ^ [(k2, v2)];

- reduce - функція приймає проміжний ключ k2 і набір значень відповідних йому [v2] і повертає значення v3: reduce :: k2 ^ [v2] ^ v3. У функціональному вигляді роботу.

При цьому функція reduce повинна реалізовувати функцію облікового гомоморфизма. Існує досить багато завдань, де дані умови виконуються:

- розподілений пошук слів по регулярному виразу (Distributed GREP);

- підрахунок частоти звернення до URL адресами (URL Access Frequency);

- пошук сторінок, що посилаються на певну посилання (ReverseWeb-LinkGraph);

- індексування Web сторінок для кожного ключового слова (Inverted Index) - розподілена сортування документів (Distributed Sort) і ін. Модель програмування MapReduce успішно застосовується і до деяких алгоритмів аналізу даних. Що використовують дану модель платформи Apache Spark і Apache Hadoop використовуються для обробки Великих даних в системах компаній Google, Yandex та ін. Модель MapReduce використовується для розподіленого виконання алгоритмів аналізу даних на багатоядерних системах. Раніше описаний підхід, декомпозіруется алгоритми аналізу даних на функції map і reduce. Однак він застосуємо тільки до алгоритмів, відповідних моделі статистичних запитів (SQM - Statistical Query Model) і обчислюють статистичні характеристики в даних або градієнт.

Основна ідея методу полягає в поділі алгоритму на дві функції: map, яка обчислює статистику для кожної порції даних і reduce, яка агрегує отримані результати. Проводяться також дослідження з адаптації різних алгоритмів аналізу до моделі MapReduce: алгоритмів кластеризації, регресійного аналізу, дерев рішень, класифікації пошуку частих наборів та ін. Також можна виділити різні сучасні бібліотеки розподілених алгоритмів аналізу даних, адаптованих до моделі MapReduce.

Apache Mahout - бібліотека алгоритмів data mining, адаптованих до моделі MapReduce, які можуть виконуватися на платформі Apache Hadoop. Вона містить ряд алгоритмів для розподіленого виконання: класифікації (Naive Bayes, Random Forest, Multilayer perceptron classifier (MLPC)), регресії (Ordinary Least Squares (OLS), Linear Regression, Support Vector Machine (SVM)), кластеризації (Canopy-clustering) . Вона також дозволяє додавати власні алгоритми, декомпозіровать на функції map і reduce. Apache Spark Machine Learning Library (MLlib) бібліотека алгоритмів machine learning, адаптованих до моделі MapReduce, які можуть виконуватися на платформі Apache Spark.

Вона містить такі алгоритми: класифікації (Logistic Regression, Random Forest, MLPC, Naive Bayes), регресії (Linear Regression, SVM), кластеризації (K-Means), пошуку частих наборів (FP-Growth). Користувач також може розширювати цю бібліотеку. ML Grid з Apache Ignite 2.0 містить алгоритми machine learning для розподіленого виконання. Вона включає в себе наступні алгоритми: Linear Regression, K-Means, MLPC, Fuzzy C-Means і k-NN (k-nearest neighbors). Вони можуть виконуватися на платформі Ignite Compute Grid, яка також реалізує модель MapReduce.

Бібліотека також є розширюваної. Для аналізу потокових Великих даних може бути використана платформа Scalable Advanced Massive Online Analysis (SAMOA) [168-170]. Вона включає в себе розподілені data mining алгоритми: Vertical Hoeffding Tree (VHT), Very Fast Decision Tree (VFDT), CluStream, Adaptive Model Rules (AMRules), PARMA.

SAMOA має адаптери для виконання алгоритмів на різних платформах: Apache Storm, Apache S4 (Simple Scalable Streaming System) і Apache Samza. Всі ці платформи реалізують модель MapReduce. SAMOA дозволяє розробникам інтегрувати існуючі алгоритми з бібліотеки MOA. Vowpal Wabbit (VW) бібліотека алгоритмів аналізу, розпочата як відкритий проект, компанією Yahoo !. В даний час вона розвивається компанією Microsoft. Вона використовує платформу Apache Hadoop для масштабованих обчислень і містить кілька алгоритмів data mining для класифікації і регресії: OLS, Matrix factorization (sparse matrix SVD), Single layer neural net (with user specified hidden layer node count), Searn (Search and Learn), Latent Dirichlet Allocation (LDA), Stagewise polynomial approximation, One- against-all (OAA), Weighted all pairs, Contextual-bandit.

Таким чином, всі сучасні бібліотеки алгоритмів аналізу Великих даних використовують різні платформи розподілених обчислень, заснованих на моделі MapReduce. Однак, адаптація алгоритму до даної моделі достатня трудомістка.

Наприклад, для перетворення типового алгоритму побудови дерев рішення C 4.5 до моделі MapReduce в роботі були додані декілька нових структур даних, а в алгоритм були додані функції підготовки даних, вибору даних і поновлення результату. Таким чином, був створений практично новий алгоритм для адаптації до моделі MapReduce.

Ще однією проблемою використання моделі MapReduce для алгоритмів аналізу є необхідність наявності властивості облікового гомоморфизма. Можна виділити наступні недоліки моделі MapReduce, що обмежують її застосування для розпаралелювання алгоритмів аналізу даних [25]:

- в повному обсязі функції аналізу даних мають властивості облікового гомоморфизма;

- для реалізації моделі, необхідна явна реструктуризація алгоритму і його поділ на функції map і reduce;

- алгоритм може бути розділений для паралельного виконання тільки в

одному місці;

- припускає розпаралелювання тільки за даними. У загальному випадку алгоритми аналізу даних часто не обмежується одним проходом по набору даних. Крім того, вони мають цикли не тільки по набору даних, але і по іншим наборам, в тому числі що будується в процесі виконання (набору правил, кластерам, класам, частим наборів і т.п.). Це дозволяє ефективно распараллеливать їх не тільки по набору даних. Наслідком складності адаптації алгоритмів є невелика кількість алгоритмів аналізу даних, реалізованих в описаних бібліотеках.

1.3 Аналіз даних з використанням хмарних технологій

З огляду на бурхливе зростання технологій розподілених обчислень і хмарних обчислювальних середовищ, природним є інтеграція технологій аналізу даних, розподілених і хмарних обчислень. В даний час існують системи, які можна було б віднести до хмарних технологій аналізу даних. Одним з перших в цій області почав працювати Китайський мобільний інститут. У 2007 в ньому почалися дослідження і розробки в області хмарних обчислень. У 2009 році він офіційно анонсував платформу для хмарних обчислень BigCloud, що включає в себе інструменти для паралельного виконання алгоритмів Data Mining Big Cloud- Parallel Data Mining (BC-PDM). BC-PDM є SaaS платформу, побудовану на базі Apache Hadoop.

Користувачі можуть завантажувати дані в сховищі (розміщене в хмарі) з різних джерел і застосовувати до них різні додатки з управління даними, аналізу даних і бізнес додатки. До складу додатків аналізу входять паралельні додатки виконують: ETL обробку, аналіз соціальних мереж, аналіз текстів (Text Mining), аналіз даних (Data Mining), статистичний аналіз.

Azure Machine Learning (Azure ML) - SaaS хмарний сервіс від компанії Microsoft Inc. Він був запущений в лютому 2015 року. Azure ML забезпечує платні сервіси, які дозволяють користувачам виконувати повний цикл

аналізу: збір даних, підготовку до аналізу, настройку, аналіз даних і оцінку результатів. Сервіс орієнтований на користувачів зі знаннями в області аналізу даних. Процес аналізу описується в вигляді графа робіт (workflow). Кожен вузол графа є модулем, який виконує певну підзадачу процесу аналізу (читання, перетворення, аналіз і т.п.). Кожен модуль може виконуватися на окремому вузлі, таким чином розпаралелювання в цьому випадку виконується тільки на рівні процесу аналізу. А

Zure ML може виконувати аналіз даних, які заздалегідь розміщені в хмарі. Для цього воно містить різні засоби для імпорту даних. Для аналізу користувач може використовувати тільки заздалегідь задані алгоритми.

У квітні 2015 року компанія Amazon випустила на ринок своє рішення для хмарного аналізу даних Amazon Machine Learning (Amazon ML). Воно являє собою сервіс для навчання Предсказательная моделей [179]. Сервіс забезпечує всі необхідні стадії аналізу: підготовку даних, побудова моделі, настройку, оцінку моделі та ін. Для використання Amazon ML користувачеві не потрібні спеціальні знання в галузі аналізу. Amazon ML вирішує тільки завдання класифікації і регресії. Він включає в себе алгоритми: binary classification, multiclass classification, and regression.

Нові алгоритми не можуть бути додані користувачем самостійно. Як і у випадку з Azure ML, сервіс Amazon ML дозволяє аналізувати дані, розміщені тільки всередині хмари. Як збільшувати або зменшувати обчислень використовується Apache Hadoop. Компанія Google в березні 2016 року представила свою платформу Cloud Machine Learning (Cloud ML), яка використовується для аналізу фотографій, перекладів і пошти. Вона використовує алгоритми машинного навчання - нейронні мережі. Даний сервіс Google забезпечує REST API для розпізнавання образів, мови, мовних перекладів і т.п. Сервіс також не дозволяє розширювати склад алгоритмів аналізу. На початку 2016 року компанія IBM випустила інтелектуальний аналітичний сервіс Watson Analytics. Він вирішує високорівневі аналітичні завдання, взаємодіючи з користувачами за допомогою запитів на природній

мові. Watson Analytics аналізує дані, розміщені в хмарі. Користувачі можуть використовувати тільки ті алгоритми і методи, які вже закладені в сервіс.

Основними недоліками описаних систем є (таблиця 1.2):

а) необхідність зберігання даних, що аналізуються всередині хмари, що в свою чергу має ряд недоліків:

- вимагає додаткових апаратних засобів для зберігання даних;

- для обробки актуальних даних потрібно або завжди зберігати їх у внутрішньому сховище або вирішувати завдання їх синхронізації з джерелом інформації;

- при завантаженні інформації виробляється перетворення у внутрішні формати, що може призвести до спотворення інформації і / або викликати помилку при завантаженні;

- забезпечення конфіденційності зберігається у внутрішньому сховище інформації лягає на провайдера сервісу, що не завжди може задовольнити власника інформації;

- не використовує переваг роботи з загальною пам'яттю.

б) прив'язка тільки до однієї технології виконання розподілених обчислень (в основному Map Reduce і її реалізація на Apache Hadoop), кожна з яких має свої недоліки і може ефективно застосовуватися тільки за певних умов;

в) рішення кінцевих бізнес-задач, а не окремих завдань аналізу даних, що обмежує можливості щодо застосування.

1.4 Аналіз розподілених даних

Необхідність аналізу розподілених даних виникає не тільки в системах Інтернету речей, але і в інших завданнях, коли джерела даних або територіально виділені один від одного або належать різним власникам. У таких системах аналізу піддаються розподілені дані.

При цьому, для виконання обчисленні можуть використовуватися як

вузли, що зберігають дані, так і вузли без даних. Дані в розподіленої середовищі можуть зберігатися по-різному:

- централізовано - всі дані зберігаються на одному вузлі;

- розподілено - дані зберігаються на різних вузлах, але пов'язані між собою, при цьому розподіл може бути двох видів (рисунок 1.5):

- горизонтальним;

- вертикальним.

При вертикальному розподілі атрибути однієї і тієї ж сутності зберігаються на різних вузлах. Типовим прикладом є зберігання інформації про одну людину в різних інстанціях (у податковій інспекції про доходи та витрати, в митних органах про вантажі, що перевозяться, в медичних органах про історію хвороб і т.п.). При горизонтальному розподілі дані про різні сутності зберігаються в базах, які мають однакову структуру (тобто набір атрибутів в кожній базі однаковий). Прикладом такого сховища є бази даних супермаркетів, які перебувають в різних регіонах і мають власні сховища даних.

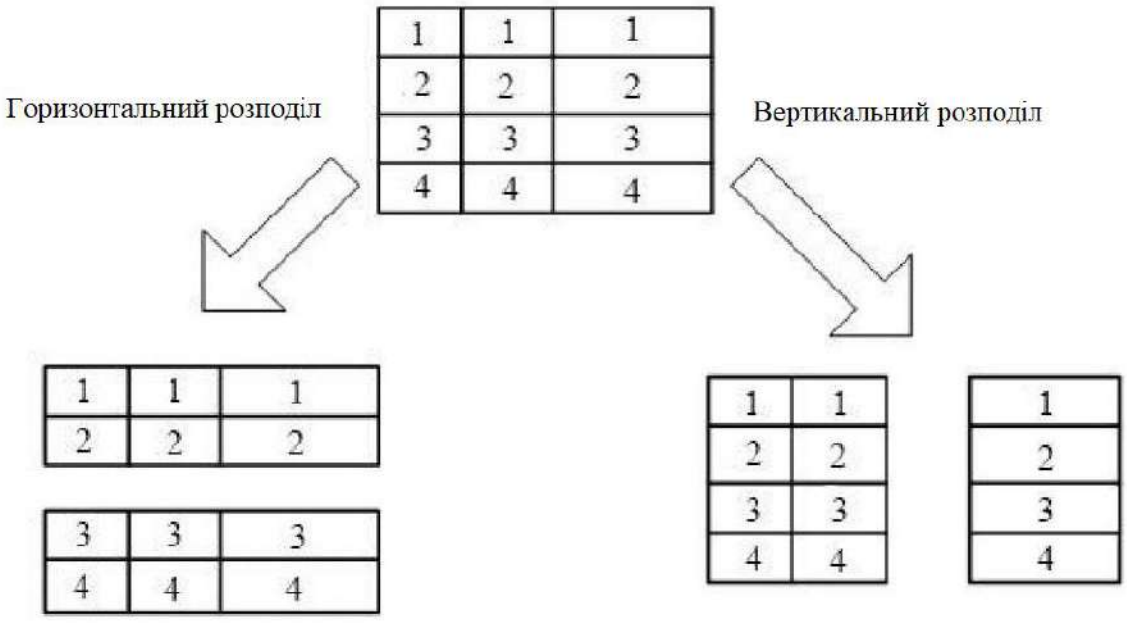

Рисунок 1.5 – Види розподілу даних

Описані вище алгоритми аналізу даних і системи їх виконують призначені для роботи тільки з централізованим зберіганням даних. В цьому випадку для розподілених джерел необхідне попереднє збір даних в єдиному сховищі, що призводить до наступних проблем: - витрачається час на передачу інформації, що може бути критично при її обробці, наприклад, в режимі реального часу; - збільшується мережевий трафік, що знижує можливості використання каналів зв'язку з низькою пропускною здатністю (супутникових каналів зв'язку, бездротових каналів і т.п.); - потрібна передача інформації, в тому числі конфіденційної, по відкритих каналах передачі даних; - підвищується цінність зібраної в одному місці інформації, що вимагає підвищених заходів забезпечення її безпеки та надійності. Уникнути зазначених недоліків дозволяє реалізація технології туманних обчислень, що припускає виконання аналізу безпосередньо на джерелі інформації або близько до нього. Однак для цього потрібна побудова паралельних алгоритмів аналізу даних з урахуванням розміщення джерел інформації для виконання функцій алгоритмів в розподіленої середовищі.

2 МЕТОДИ ПОБУДОВИ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ З ВИКОРИСТАННЯМ ШНМ

2.1 Етапи побудови ПА

Паралельний алгоритм - алгоритм, який може бути реалізований по частинах на безлічі різних обчислювальних пристроїв з наступним об'єднанням отриманих результатів і отриманням коректного результату. При розробці послідовних алгоритмів прийнято вважати, що виконується п'ять етапів:

а) постановка завдання;

б) створення математичної моделі;

в) розробка алгоритму рішення в рамках створеної математичної моделі;

г) написання програми, що реалізує розроблений алгоритм;

д) виконання програми.

При побудові паралельних алгоритмів в цій послідовності з'являються додаткові етапи:

- постановка завдання;
- створення математичної моделі;
- розробка алгоритму;
- декомпозиція алгоритму (decomposition);
- визначення незалежних блоків;
- вибір програмної моделі;
- реалізація паралельного алгоритму;
- розміщення алгоритму на виконавців (mapping);
- виконання програми.

Додаються кроки є досить трудомісткими для розробки паралельних алгоритмів. Для кожного з них проводяться окремі дослідження і існують

різні методи їх виконання. Розглянемо кожен з кроків більш докладно.

Декомпозиція алгоритму ділить алгоритм на блоки, що складаються з підмножин операцій, які повинні бути виконані спільно і не можуть бути розділені. Деякі з таких блоків можуть бути виконані в довільному порядку, отже, і паралельно на різних виконавців. Така декомпозиція можлива не завжди. Існують алгоритми, які принципово не допускають при своїй реалізації участі декількох виконавців.

Після декомпозиції алгоритму на окремі блоки, необхідно визначити які з них можуть виконуватися незалежно один від одного. Виділяють наступні види залежностей:

- залежність по управлінню, яка визначає порядок команд по відношенню до команди умовного переходу, тобто команди, які не є командами переходу, виконуються тільки коли виконуються відповідні їм умови - залежність за даними, яка визначає використання одних і тих же даних різними командами; розрізняють наступні види таких залежностей:

- залежність по входу; виникає, коли одні і ті ж дані надходять на вхід кільком командам;

- потоковая залежність; виникає, коли дані є виходом однієї команди надходять на вхід наступної команди;

- антизаленість; виникає, коли дані є виходом однієї команди надходять на вхід попередньої команди;

- залежність по виходу; виникає, коли виходами декількох команд є одні й ті ж дані.

Є два обмеження, пов'язані з залежностями з управління: - залежна з управління команда, не може бути в результаті переміщення поставлена перед командою умовного переходу, стати незалежною від нього; - команда, яка не залежить з управління від команди умовного переходу, не може бути поставлена після команди умовного переходу так, що її виконання стане управлятися цим умовним переходом. У частині залежності за даними можуть бути встановлені обмеження сформульовані Бернстайном:

- відсутність антізавісімості;
- відсутність потокової залежності;
- відсутність залежності по виходу.

Вони є достатніми, але не необхідними.

2.2 Моделі паралельного програмування

Вибір моделі паралельного виконання алгоритму визначається його блоками, які будуть виконуватися паралельно, схемами взаємодії між ними, а також середовищем виконання. Виділяють наступні моделі для паралельного виконання: - модель передачі повідомлень. Припускає, що працює додаток складається з набору процесів з різними адресними просторами, кожен з яких функціонує на своєму виконавця.

Процеси обмінюються даними за допомогою передачі повідомлень через явні операції send/receive. Перевага моделі полягає в тому, що програміст здійснює повний контроль над вирішенням завдання, недолік - в складності програмування - модель розділяється пам'яті.

Припускається, що додаток складається з набору потоків виконання, що використовують колективні змінні і примітиви синхронізації. - модель розділених даних. Припускається, що додаток складається з наборів процесів або потоків, кожен з яких працює зі своїм набором даних, обміну інформацією при роботі немає. Така модель може бути застосована до обмеженого класу задач.

Перша модель добре переноситься, дає повний контроль над виконанням, але дуже трудомістка.

Друга модель легка для програмування, але не дає можливості повністю контролювати рішення задачі.

Третя модель може бути застосована при декомпозиції за даними і в разі якщо результати обробки окремих частин даних можуть бути об'єднані (у випадках, якщо використана декомпозиція за даними: розділяй і

володарюй).

2.3 Реалізація паралельних алгоритмів

На етапі програмування необхідно здійснити вибір мови програмування і засобів виконання алгоритму. Мови програмування з точки зору підтримки паралельних обчислень можна розділити на наступні класи:

- мови без підтримки паралельних обчислень (C/C ++, Паскаль, Фортран та ін.). Такі мови практично не можуть використовуватися для реалізації паралельних алгоритмів або вимагають додаткових програмних засобів, що реалізують паралельні обчислення;

- мови, що мають підтримку параллельнх обчислень на рівні синтаксису (Ада, High Performance Fortran та ін.) і директив компілятора (OpenMP та ін.) Дозволяють розпаралелювати тільки окремі структури, наприклад цикли;

- мови, що мають підтримку паралельних обчислень на рівні програмних розширень (Java, C # та інші): бібліотек, класів та ін .;

- мови, структурно підтримують паралелізм (функціональні мови програмування Lisp, Haskell та ін.) І передбачають побудову програм з структур, які можуть бути виконані паралельно.

Перші три групи мов відносяться до імперативним мовам програмування. Вони орієнтовані на роботу відповідно до моделі машини Тьюринга. Її основною ідеєю є зміна стану машини Тьюринга і під час кожної інструкції програми. Таким чином, програми, написані на імперативних мовах програмування, припускають наявність стану програми і його зміна в процесі виконання.

Їх основною проблемою при паралельному виконанні є необхідність забезпечення одночасного доступу до стану програми з паралельних гілок. Це породжує проблеми синхронізації доступу, блокування, гонки та ін. Їх рішення досить трудомісткий процес, як при розробці, так і при

налагодженні. На відміну від них функціональні мови засновані на теорії Х-обчислень запропонованої Алонсо Черчем одночасно з Тьюрингом. Однак на відміну від машини Тьюринга, в теорії Х-обчислень програма представляється у вигляді функціонального вираження, в якому функції викликаються одна з одної. Вся інформація необхідна для передачі між функціями передається через їхні аргументи і повертаються значення. Таким чином, програми, написані на функціональних мовами, не використовують внутрішній стан.

Функції, з яких будується функціональна програма, є чистими. Отже, такі функції можуть виконуватися паралельно без необхідності додаткових заходів запобігають блокування, гонки і інші проблеми паралельного виконання. Теоретично функціональні вирази, побудовані з таких функцій, можуть бути розпаралелені автоматично.

2.4 Роробка методу аналізу даних

Вихідною інформацією для алгоритмів аналізу даних є набори даних, що аналізуються. Результатом є модель знань, що будується алгоритмом і містить витягнуті з даних закономірності. Для паралельного виконання алгоритм повинен бути декомпозирован на блоки, які можуть бути виконані паралельно. Для алгоритмів аналізу даних такими блоками є функції, які виконують обробку як вихідних даних, так і моделі знань, що будується алгоритмом. Паралельне виконання таких функцій означає можливість їх виконання в довільному порядку.

Прикладом вертикально розподілених даних є безліч датчиків, що вимірюють різні характеристики одних і тих же об'єктів (наприклад, кінематичні параметри, що збираються датчиками, встановленими вздовж траси одних і тих же об'єктів, що рухаються). Прикладом, горизонтально розподілених джерел даних є безліч однотипних датчиків, що вимірюють показники різних об'єктів (наприклад, датчики неруйнівного контролю,

встановлені на різних будівлях міста).

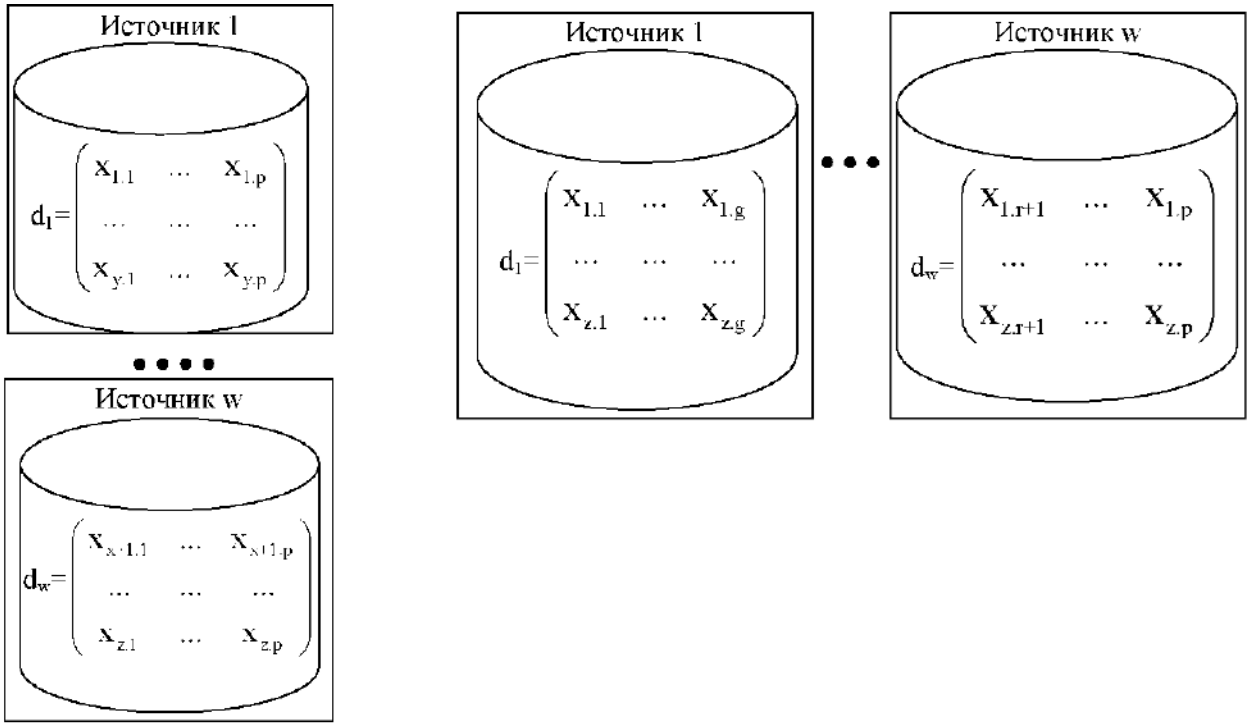

Рисунок 2.1 – Розподіл даних

2.5 Модель представлення знань

Модель знань містить закономірності, витягнуті з набору даних в процесі їх аналізу. Такі закономірності описуються: математичними функціями, класифікаційними правилами, центрами кластерів і т.п. Формально елемент моделі знань (закономірність) можна уявити кортежем параметрів різного типу.

В процесі виконання алгоритму аналізу даних над елементами моделі знань виконуються операції: пошуку (вилучення), вставки (додавання), видалення [4]. Залежно від алгоритму аналізу, ті чи інші операції можуть виконуватися частіше або рідше, проте вони будуть виконуватися всі. Допустивши, що кожна операція в процесі роботи алгоритму в середньому виконується приблизно однакову кількість разів, все елементи моделі знань

будуть організовані в деревоподібну структуру, для якої час виконання всіх операцій має логарифмічну залежність від розміру.

При цьому елементи моделі знань, що є дочірніми елементами одного і того ж елемента, повинні мати однаковий набір властивостей. Таким чином, модель знань являє собою ліс дерев, що може бути представлений як масив (індексований список) кореневих елементів дерев [2, 7]:

$$m = [e_0, e_1, e_2, ..., e_v,..., e_w]$$

Склад масиву і порядок проходження елементів в ньому строго визначений типом моделі і не змінюється в процесі роботи алгоритму аналізу даних. Наприклад, стандарт PMML версії 4.2 визначає 15 типів моделей знань. Крім того, склад також залежить від особливостей алгоритму аналізу даних, який її будує. При застосуванні побудованої моделі знань до нових даних, необхідно визначати чи відповідає вона цими даними.

Для цього модель знань повинна зберігати інформацію про дані (метадані) за якими вона була побудована. Так стандарт PMML для цих цілей визначає тег MiningSchema, в якому перераховуються атрибути, використовувані в моделі знань, і їх характеристики. Введемо до складу моделі знань елементи, що описують метадані набору аналізованих даних. Вони включають в себе опис джерела (наприклад, uri до файлу), опис атрибутів і їх значень:

e0= <[curr, size, uri, login, password, ...], [e0.1, e0.2, ..., e0.k, ..., e0.p]>,

де e0.k елемент моделі знань, що містить інформацію (номері, імені, ролі і його можливих значеннях) про k-м атрибуті ak:

e0.k= <[pos, size, name, role ....], [vk.1, ..., vk.q, ..., vk.u]>

Таким чином, модель знань можна представити у вигляді лісу дерев, перший з яких, є обов'язковим і містить інформацію про аналізованих даних (рисунок 2.2).

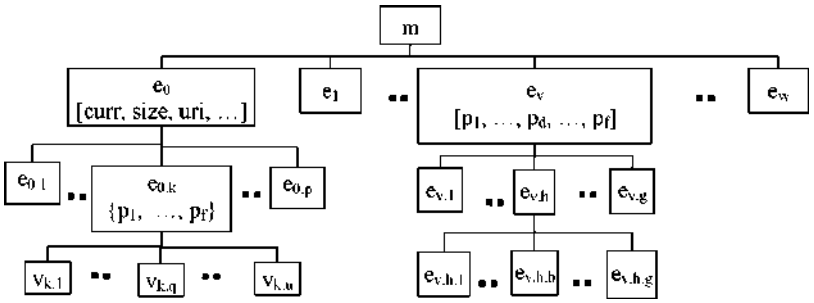

Рисунок 2.2 – Подання моделі знань у вигляді лісу дерев її елементів

Для індексації елементів моделі знань використовуємо децимальних нотацію індексів елементів лісу дерев. Для зберігання і обробки індексів в децимальних нотації використовуємо список цілих чисел. Перше число індексу означатиме кореневий елемент відповідного дерева в моделі знань m [v] = ev. Наступні числа в індексі будуть позначати позиції елементів моделі знань серед вузлів-нащадків дерева. Наприклад, елемент моделі знань m [2, 1, 3] позначає 3й елемент моделі знань серед вузлів-потоків 1 -го вузла серед вузлів-нащадків кореневого вузла 2-го дерева моделі знань m. Всі безліч індексів моделі знань позначимо символом I.

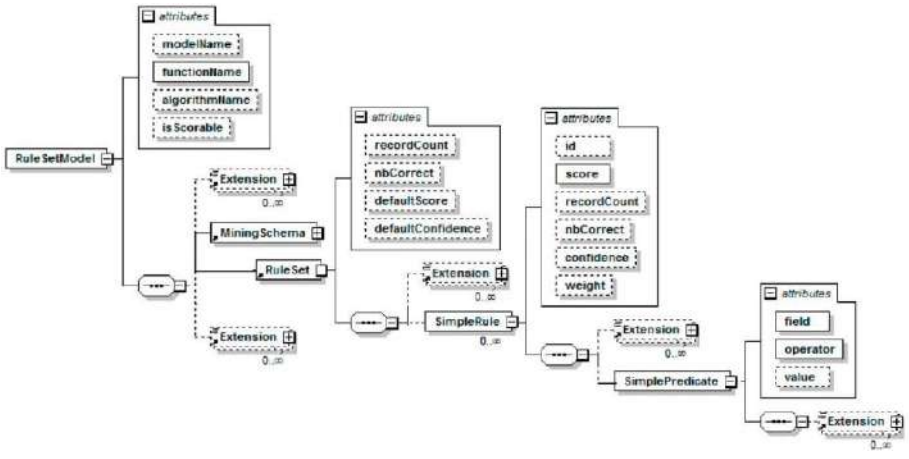

Рисунок 2.3 – PMML модель знань RuleSetModel

2.6 Модель та алгоритми кластеризації з використанням модифікованих ШНМ типу карт Кохонена

Базовим алгоритмом кластеризації на основі нейронних мереж Кохонена це самоорганізування карти. Модель роботи систем, що самоорганізують карти Кохонена представлена на рисунку 2.1.

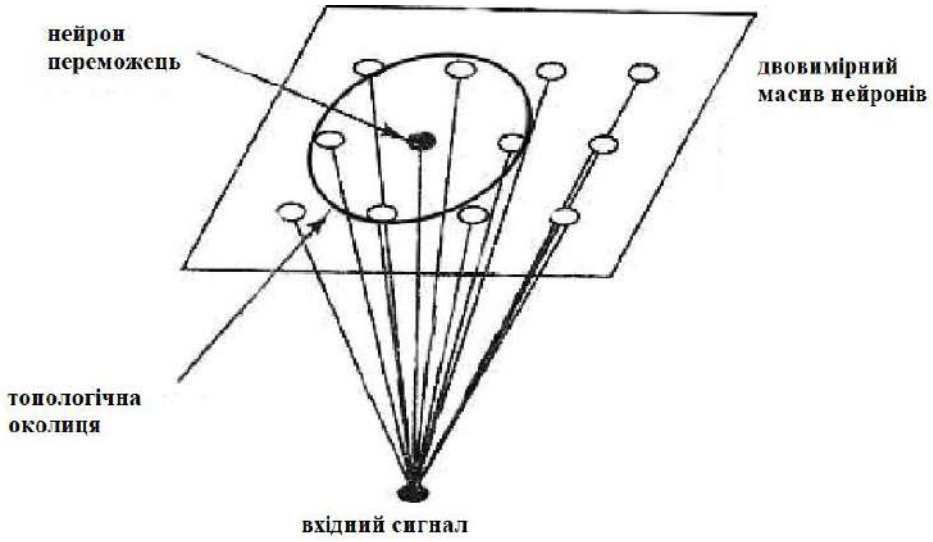

Рисунок 2.4 – Модель SOM

Шар Кохонена складається з n паралельно діючих лінійних елементів. З однаковим числом входів m, отримують на входи один і той же вектор вхідних сигналів x=(xb...xm). На виході j-го лінійного елемента отримуємо сигнал:

$$y_i = W_{j0} + \sum_{i=1}^{m} W_{ji}\,, x_i$$

де Wij - ваговий коефіцієнт i-го входу j-го нейрона, Wj0 - пороговий коефіцієнт.

Після проходження лінійних суматорів сигнали посилаються на обробку за правилом winner takes all (WTA) або winner takes major (WTM).

Серед вихідних сигналів yj вибирається максимальний:

$$j_{max} = argmax\{y_j\}$$

Для правила WTA на виході сигнал с jmax дорівнює одиниці, решта – нулю. В випадку, коли кілька визначається для декількох jmax, то в залежності від ситуації або прийнятого алгоритму встановлюється одиниця всім або першого в списку.

Базовими представниками алгоритмів кластеризації для WTA є SOM, для WTM – GNG.

Набір даних d є D зазвичай містить характеристики (температура, рівень шуму, вібрація, тиск і тд.) Об'єктів (наприклад, блоків складних систем, транспорту і тд.). Ми представляємо набір даних у вигляді двовимірного масиву, наприклад, для т об'єктів, які описані p характеристиками]:

$$d = (x_{j.k})_{i=1.k=1}^{m,p} \quad (2.1)$$

де $x_{j.k}$ значень k-ой характеристики об'єкта.. У разі розподіленого зберігання дані розділені між вузлами і представлені як частини матриці даних d:

$$d = d_1 \cup d_2 \cup \ldots \cup d_h \cup \ldots \cup d_w \quad (2.2)$$

де підматриці даних на d_h на h.

В такому випадку, можливі наступні варіанти розподілених даних:

горизонтальний розподіл:

$$d_1 = (x_{j.k})_{j=1,k=1}^{y,p}, d_2 = (x_{j.k})_{j=y+1,k=1}^{x,p}, \dots, d_w = (x_{j.k})_{j=x+1,k=1}^{z,p} \quad (2.3)$$

вертикальне розподілення:

$$d_1 = (x_{j.k})_{j=1,k=1}^{z,g}, d_2 = (x_{j.k})_{j=1,k=g+1}^{z,r}, \dots, d_w = (x_{j.k})_{j=1,k=r+1}^{z,p} \quad (2.4)$$

Прикладом вертикально розподілених даних (рисунок 2.3 а) є набір датчиків, що вимірюють різні характеристики одних і тих же об'єктів (наприклад, кінематичні параметри, зібрані датчиками, встановленими вздовж маршруту одних і тих же об'єктів, що рухаються). Прикладом горизонтально розподіленого джерела даних (рисунок 2.3б) є набір датчиків одного типу, які вимірюють характеристики різних об'єктів (наприклад, датчики неруйнівного контролю, встановлені в різних міських будівлях).

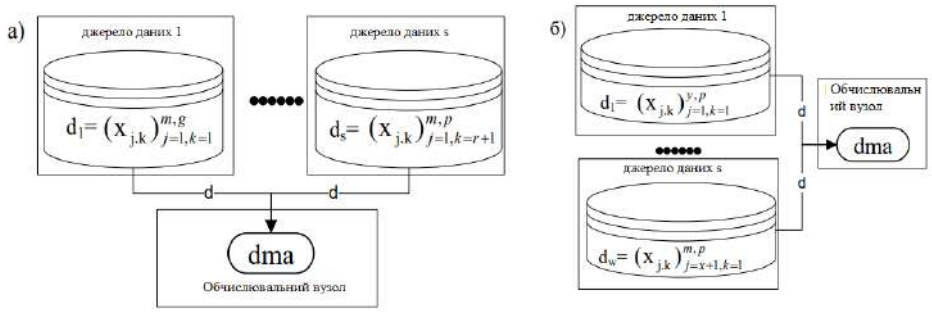

Рисунок 2.5 – Обробка розподілених даних: а) вертикальна; б) горизонтальна

SOM представляє собою масив нейронів:

$$\mu = [n1, \dots, nn], \quad (2.5)$$

де n - число нейронів, встановлені аналітиком. Кожен нейрон n_i визначається вектором моделі n_i з вагами ωk:

$$ni = [\omega 1, \ldots, \omega p];$$

Алгоритм SOM формально можливо уявити як композицію функції:

$$som = f_n \circ f_{n-1} \circ \ldots \circ f_1 \circ f_0 \qquad (2.6)$$

де ° - оператор композиції, котрий застосовується справа наліво.

В (2.6), функция f0: D → M приймає данні d ∈ D як аргумент и вертає SOM μ0 ∈ M. Наступна функція, ft, t=1,..,n приймає SOM μt-1 ∈ M створенний попердньою функцією і вертає SOM μt ∈ M з зміненими вагами:

$$ft : M \rightarrow M \qquad (2.7)$$

Щоб застосувати деяку функцію f_t до кожного елементу матриці даних a, ми викликаємо її в циклі. Ми вводимо цикли над стовпцями і рядками матриці даних для итеративной обробки даних:

loopc застосовує функцію f_t к столбцям матриці даних d ∈ D, починая з індекса i_s до індекса i_e:

loopc: I → I → (M → M) → D →M → M (2.8)

loopc i_s i_e f_t d μ = (ft d[*, i_e])°…°(f_t d[*, i_s]) μ

loopr применяет функцию f_t к строкам матрицы данных d ∈ D начиная с индекса i_sдо индекса i_e:

loopr: I → I → (M → M) → D → M → M (2.9)

loopr i_s i_e f_t d μ = (f_t d[i_e, *])°…°(f_t d[i_s, *]) μ

A loop loopr застосовує функцію f_t к нейронам SOM μ починаючи з індекса i_s до індекса i_e:

loopn: I → I → (M → M) → M → M (2.10)

loopn i_s i_e f_t μ = (f_t μ[i_e])°…°(f_t μ[i_s])

Функція вищого порядку Paralleld висловлює паралельне виконання функцій на розподілених вузлах:

$$\text{paralleld} : [(M \rightarrow M)] \rightarrow M \rightarrow M$$

$$\text{paralleld}\ [f_r, \ldots, f_s]\ \mu = \text{join}\ \mu\ (\text{forkd}\ [f_r, \ldots, f_s]\ \mu), \qquad (2.11)$$

де функція *forkd* дозволяє викликати функції паралельно:

$$\text{forkd} : [M \rightarrow M] \rightarrow M \rightarrow [M]$$

$$\text{forkd}\ [f_r, \ldots, f_s]\ \mu = [f_r\ (\text{copy}\ \mu), \ldots, f_s\ (\text{copy}\ \mu)]; \qquad (2.12)$$

і функція *copy* створює копії SOM μ в розподілених вузлах для паралельної обробки функціями:

$$\text{copy}: M \rightarrow M,\ \text{copy}\ \mu = [\mu_1, \mu_2, \ldots, \mu_s]. \qquad (2.13)$$

Функція *join* об'єднує карти SOM, побудовані паралельними функціями в розподілених вузлах:

$$\text{join}: M \rightarrow [M] \rightarrow M. \qquad (2.14)$$

Реалізація функції join залежить від елементів моделі.

Модифікований алгоритм SOM будує нейронні мережі з шаром Кохонена в якості основного елемента. Для кожного вхідного рядка набору даних d в якості переможця вибирається нейрон, вага якого найбільш схожа на нього. Ваги переможця і сусідів переможця (визначаються функцією сусідства G) коригуються у напрямку до вхідних рядку з використанням швидкості навчання η. яка є позитивною постійною, визначеним користувачем. Таким чином, в результаті навчання SOM класифікує вхідні рядки набору даних в кластери.

Алгоритм SOM можна представити наступною послідовністю кроків:

Начальна ініціалізація.

Вихідний вектор ваг w_i

$w_i = \{w_{i1}, w_{i2}, \dots, w_{is}\}$

s - розмірність вхідного вектора даних, зазвичай заповнюється випадковими значеннями.

Цикл підвибірки.

Вибирається вектор вхідних даних *x*:

$x = \{x_1, x_2, \dots, x_s\}$.

Вектор *x* застосовується до ваг нейронів.

Пошук максимальної подібності.

Пошук нейрона-переможця, виконується на основі мінімальної відстані від нейрона до вхідного вектора даних:

$$W_{winner} = argmin_{i=1}^{n} \|x - w_i\|$$

де W_{winner} - нейрон переможець, n - кількість нейронів, i - поточний нейрон, x - вхідний вектор даних.

Зазвичай в якості метрики використовують Евклідова відстань:

$$d_i = \sum^{m} (w_j - x_j)^2$$

де d_i - відстань i-го нейрона до вхідного вектора даних, m - кількість атрибутів, w_{ij} - j-ий атрибут i-го нейрона, x_j- j-ий атрибут вхідного вектора даних.

Коригування ваг:

$$w_i(t+1) = (1 - kf^{a_i}) * w_i(t) + kf^{a_i} h(t) x$$

де i - поточний нейрон, t - ітерація, kf - коефіцієнт стяги, який виставляється зазвичай в районі 0.1, a_i - порядок сусідства i-го нейрона, h (t) - функція експоненціального убування для зменшення впливу нових вхідних

векторів на нові нейрони і перенавчання моделі, може бути опущена, x - вхідний вектор даних.

Формула застосовується для нейронів околиці нейрона-переможця, що задовольняють умові:

ai < neighbor_area

Для нейрона переможця форма коригування виглядає простіше через відсутність обліку околиці і більшого впливу вхідного вектора на його коригування:

$$w_{winner}(t+1) = (1 - kf)w_i(t) + kf * h(t) * x$$

Повернення до 2 кроку (вибір підвибірки), поки не буде досягнута умова зупинки алгоритму.

Опис моделі алгоритму SOM

m[0] - метадані. Зберігає інформацію про атрибути вхідних даних.

m[1] - нейрони самоорганізовують карти Кохонена, складається з *n* нейронів, кожен нейрон являє собою вектор ваг, розмірність ваг відповідає кількості атрибутів вхідного вектора даних m.

m[1, i] – 1-ий нейрон, одиниця самоорганізує карти Кохонена.

m[1, i, j].w – ja вага іго нейрона, визначає близькість нейрона до вхідного вектора за відповідним атрибуту.

m[1].i_min - індекс нейрона з найменшою відстанню до вхідного вектора X[k, j], при цьому:

m[1, m[1].i_min] - нейрон переможця для вектора X[k, j]

m[1].a - функція швидкості навчання, лінійна і обернено пропорційна кількості ітерацій. Призводить до того, що вектора вхідних даних вносять приблизно рівний внесок в результат навчання.

m[1].t - номер поточної ітерації. Впливає на коефіцієнт сусідства при

коригуванні ваг переможця. Зі збільшенням кількості ітерацій коефіцієнт сусідства зменшується .

m[2, i].r - список відстаней від поточного вхідного вектора до і-го нейрона. Сума відстаней атрибутів і ваг відповідних векторів і нейронів.

3 РЕАЛІЗАЦІЯ МОДИФІКОВАНОГО МЕТОДУ ПОБУДОВИ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ АНАЛІЗУ ДАНИХ

При побудові паралельного алгоритму аналізу данни основним завданням є пошук його оптимальної структури з точки зору можливості і ефективності паралельного виконання. Це залежить не тільки від самого алгоритму, але і від характеристик набору аналізованих даних і середовища виконання. Тому важливим є формування різних варіантів паралельного виконання.

Для паралельного виконання функцій в функціональному вираженні, відповідно до теореми Черча-Россера, вони повинні викликатися як аргументи іншої функції. Для цього необхідно визначити перетворення, що виконуються над алгоритмом аналізу даних, представленого як описано в попередньому розділі, у вигляді композиції уніфікованих функцій. Крім того, для паралельного виконання алгоритму аналізу даних декомпозіровать на ФОМ необхідно визначити які з них можуть бути виконані паралельно. Для цього повинні бути визначені інформаційні залежності між ними.

З огляду на, що ФОМ обмінюються моделями знань, інформаційні зв'язки визначаються робочою ними елементами моделей знань. Таким чином, для аналізу інформаційних залежностей і побудови паралельних алгоритмів аналізу даних необхідно визначити умови, що не порушують інформаційні залежності при яких паралельне виконання ФОМ буде коректним. На їх підставі може бути розроблений метод розпаралелювання послідовного алгоритму аналізу даних.

parallel : Boolean →[(M →M)] →M →M.parallel s [f1, ..., fr] m = if (s == true) head fork [f1, ..., fr] m else join m (fork [f1, ..., fr] m),

де head - функція, яка повертає перший елемент списку. Функція parallel використовує функцію вищого порядку fork, яка приймає список ФОМ і модель знань, застосовує кожну ФОМ зі списку до моделі знань і отриману модель знань додає в повертається список.

ork : Boolean →[M →M] →M →[M].fork s [f1, ..., fr] m= if (s == true) [f1m, ..., frm]m else[f1clone m, ..., frclone m]

У функціях parallel і fork перший булевский аргумент s визначає спосіб виконання: із загальною або розподіленою пам'яттю. Для систем з розподіленою пам'яттю, кожна паралельно виконується ФОМ працює з аргументами, що зберігаються в окремій пам'яті. Для створення окремих копій моделі знань, повинна використовуватися функція клонування clone. Паралельно побудовані моделі знань m1,., Mk, також зберігаються в окремих областях розподіленої пам'яті і повинні об'єднуватися в єдину модель знань mr (рисунок 3.1). Для цього повинна бути використана функція об'єднання join (2.10).

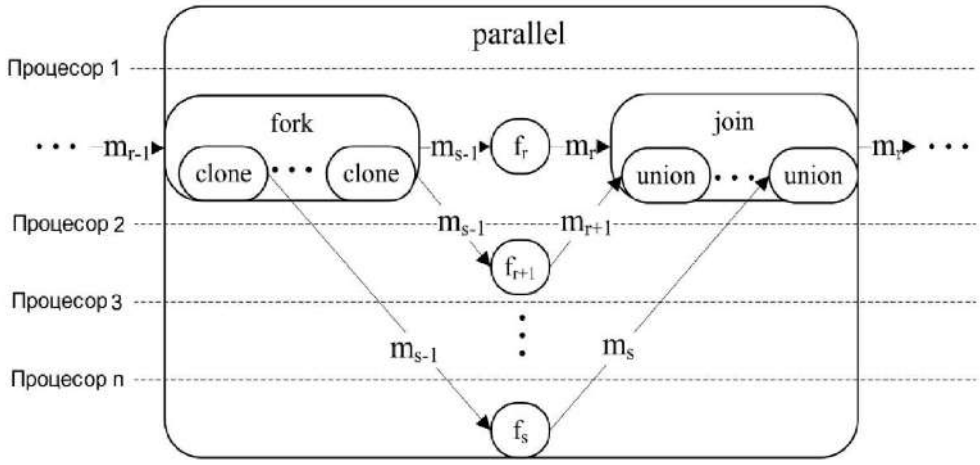

Рисунок 3.1 – Паралельне виконання для систем з розподіленою пам'яттю

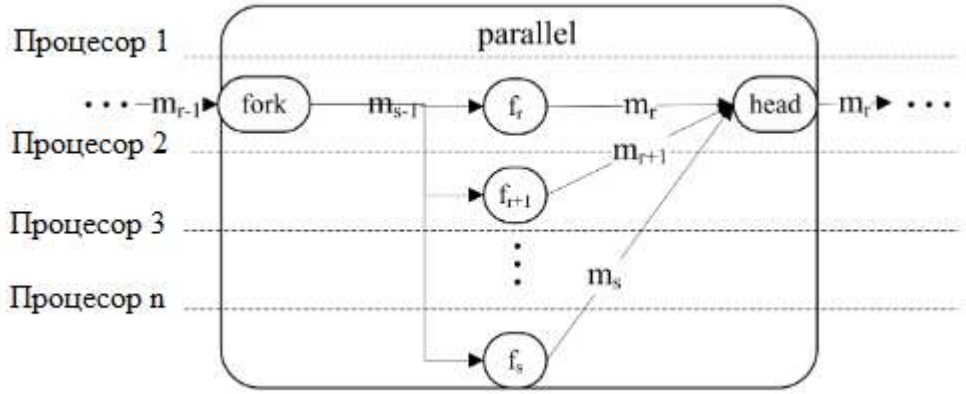

Рисунок 3.2 – Паралельне виконання для систем з загальною пам’яттю

Алгоритм аналізу даних, представленнийв вигляді композиції ФОМ (2.12), може бути перетворений в паралельну форму шляхом передачі в якості аргументу в функцію parallel (3.1) тих ФОМ, які можуть бути виконані паралельно відповідно до вищеописаними условіямі.В загальному випадку в алгоритмі аналізу даних може бути кілька місць, в яких ФОМ можуть бути виконані параллельно.Для визначення всіх можливостей паралельного виконання последовательногоалгорітма аналізу даних, з урахуванням ОЧПК, предложенметод розпаралелювання. Його псевдокод наведено нижче.

Запропонований метод здійснює обхід дерева вкладеності алгоритму даних в ширину (breadth-first search, BFS). Для кожного вузла (ФОМ) дерева вкладеності виконується перевірка можливості паралельного виконання в парі з суміжним вузлом (ФОМ, що виконується послідовно з поточним). Перевірка здійснюється за допомогою створюваного кортежу pair (рядок 4), який зберігає пари суміжних вузлів (послідовних ФОМ). Кожна пара перевіряється на можливість розпаралелювання (рядок 6). Якщо розпаралелювання можливо, то ФОМ додаються в раніше створену (рядок 10) або в новостворювану функцію parallel (рядок 14). Для створеної функції parallel обчислюється КР kparanei. Якщо він менше 1 то вузли дерева (ФОМ), що виконуються паралельно, замінюються в батьківському вузлі створеної функцією parallel, інакше розпаралелювання не виконується. Далі

обчислюється ОЧПК q. Все ФОМ групуються в q композицій. Якщо поточний вузол дерева (ФОМ) є циклом (рядок 17), то перевіряється можливість паралельного виконання функції loope, застосовуваної до різних елементів моделі знань (рядок 18). Якщо паралельне виконання ФОМ можливо, то створюється функція parallel для loope. Поточний вузол замінюється в батьківському вузлі створеної функцією parallel.

Середу виконання алгоритмів аналізу даних можна представити у вигляді спрямованого графа (рисунок 3.3):

$$Ом = E>,$$

де:

- E - безліч дуг, що з'єднують вершини і відповідні засобам передачі даних (СПД) між вузлами;

- N - безліч вершин графа, відповідні розподіленим вузлам, які в залежності від місця зберігання даних, що аналізуються можна розділити на дві підмножини: N = де - вузли джерела, які зберігають аналізовані дані.

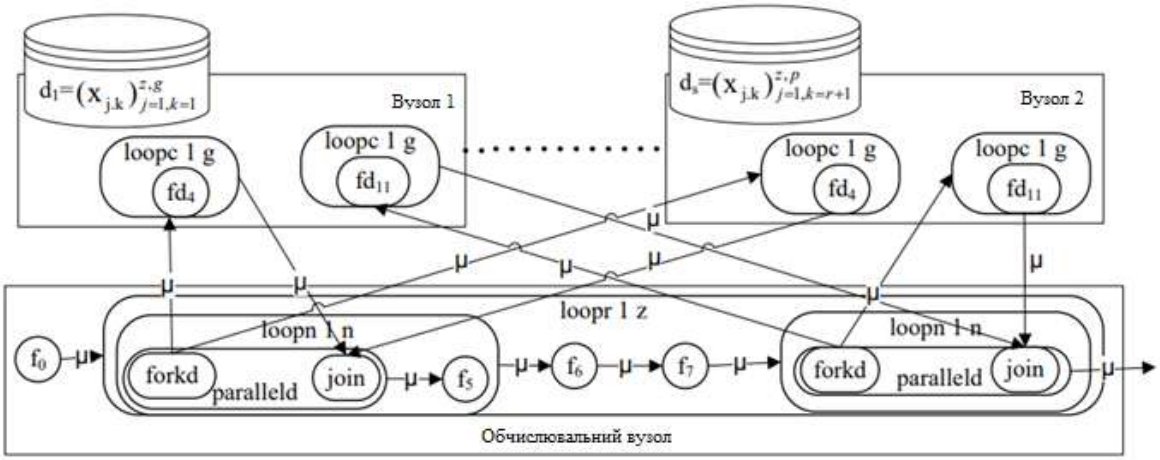

Рисунок 3.3 – Фізичне середовище виконання алгоритмів аналізу даних

З огляду на, що вузли середовища виконання можуть бути багатопроцесорними (багатоядерними), на них може виконуватися кілька потоків. Отже, будь-який вузол n_i е N можна представити як безліч процесорів:

$$\tau(f) = \tau(h_0) + {}_{0...|H|1}\max(h)_{kgiik}\tau_{==}\sum, \text{ где } h_i \in H_k.$$

Цей вираз вірно, для послідовно виконуються підмножин виконавців. Тому виконавці, які можуть виконуватися паралельно, повинні об'єднуватися в одну підмножину.

$$n_i = \{t_{i1}, t_{i2}, ..., t_{ie}, ..., t_{iz}\}$$

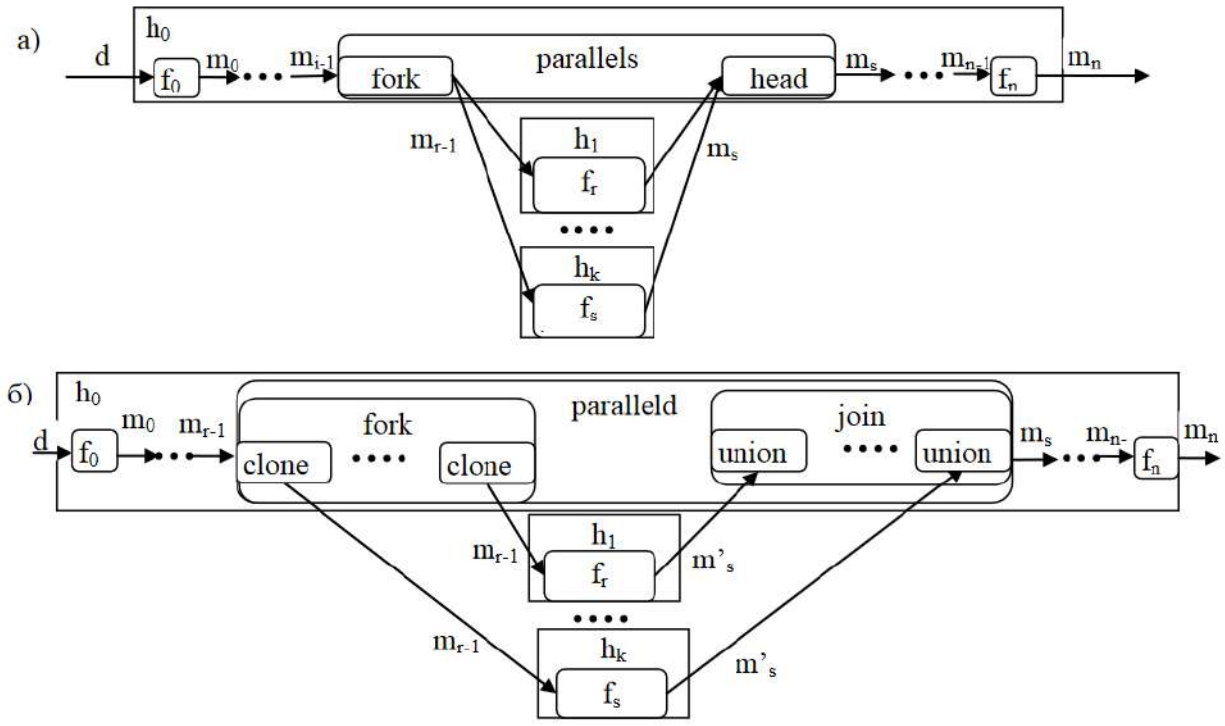

Рисунок 3.4 – Паралельне виконання алгоритму аналізу даних на виконавців (а -для функції parallels; б -для функції paralleld)

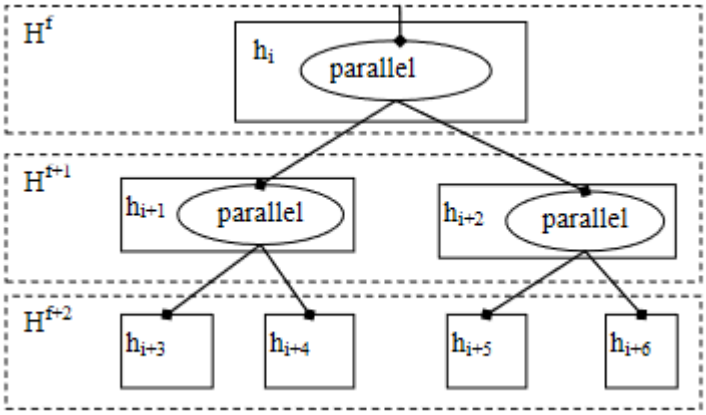

Рисунок 3.4 – Об'єднання підмножин виконавців

3.1 Реалізація бібліотеки

Для практичного застосування, отримані результати були реалізовані у вигляді каркаса бібліотеки алгоритмів аналізу даних для паралельного і розподіленого виконання [24, 25]. Для цього була розширена бібліотека Xelopes компанії PrudSys (http://www.prudsys.de/). Бібліотека реалізована на мові програмування Java. Її ядро містить базові класи для реалізації алгоритмів аналізу даних. Її основним недоліком, є монолітна реалізація алгоритмів, яка не дозволяє його розпаралелити без зміни коду. Для усунення цього недоліку, представивши алгоритм у вигляді окремих функцій алгоритму, ядро даної бібліотеки було модифіковано. Таке уявлення відповідає функціональної моделі алгоритму аналізу даних. Розширена версія отримала назву DXelopes (Distributed Xelopes).

В результаті бібліотека DXelopes, має такі можливості в порівнянні з існуючими бібліотеками алгоритмів аналізу даних:

- побудова нових алгоритмів шляхом комбінації існуючих ФОМ або модифікація існуючих алгоритмів шляхом заміни (або модифікації) окремих ФОМ;

- перетворення послідовних алгоритмів, побудованих у вигляді послідовності ФОМ, в паралельні форми для виконання в паралельних і

розподілених середовищах;

- виконання паралельних алгоритмів аналізу даних в різних обчислювальних середовищах.

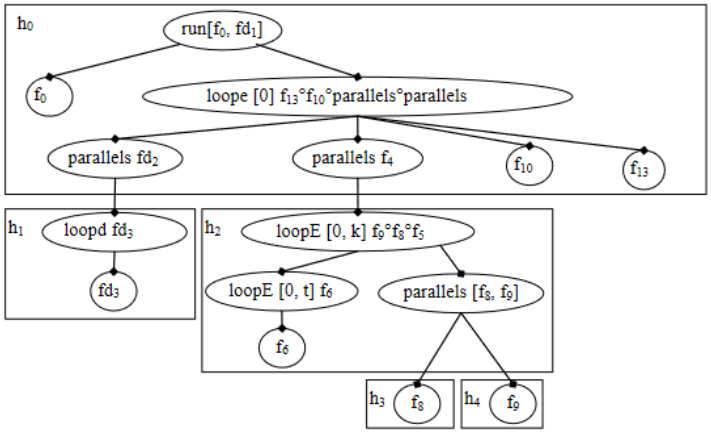

Рисунок 3.5 – Об'єднання підмножин виконавців

Розглянемо побудову дерева виконавців для алгоритму 1R представленого виразом (3.4). Відповідно до описаним методом побудови дерева виконавців (функція createExecutorTree) для основної послідовності створюється виконавець h_0, що є коренем дерева (рисунок 3.6) .У нього включаються ФОМ основної послідовності (функцією fullExecutor) композиції паралельного алгоритму 1R (3.6):

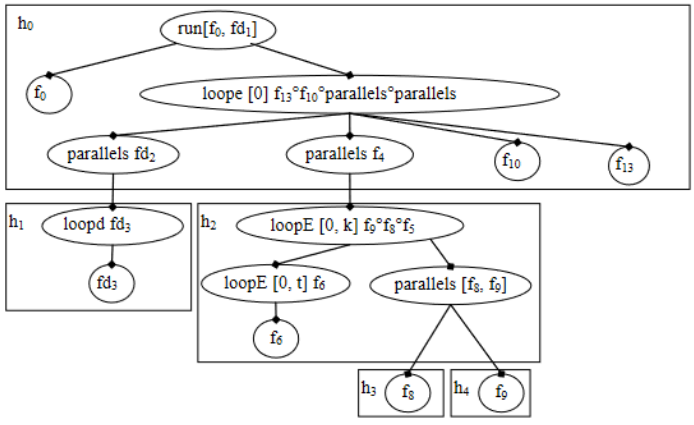

Рисунок 3.6 – Дерево виконавців для паралельного алгоритму 1R

Виконавці h_1 и h_2 включають в себе піддерева з кореневої ФОМ, що є циклом. Отже, при виконанні алгоритму аналізу даних вони можуть бути розтиражовані за кількістю оброблюваних елементів моделі знань. Виконавцы h_1, h_2, h_3і h_4 викликаються з функції для систем зі спільною пам'яттю parallels. Отже, вони можуть виконуватися в системах із загальною пам'яттю.

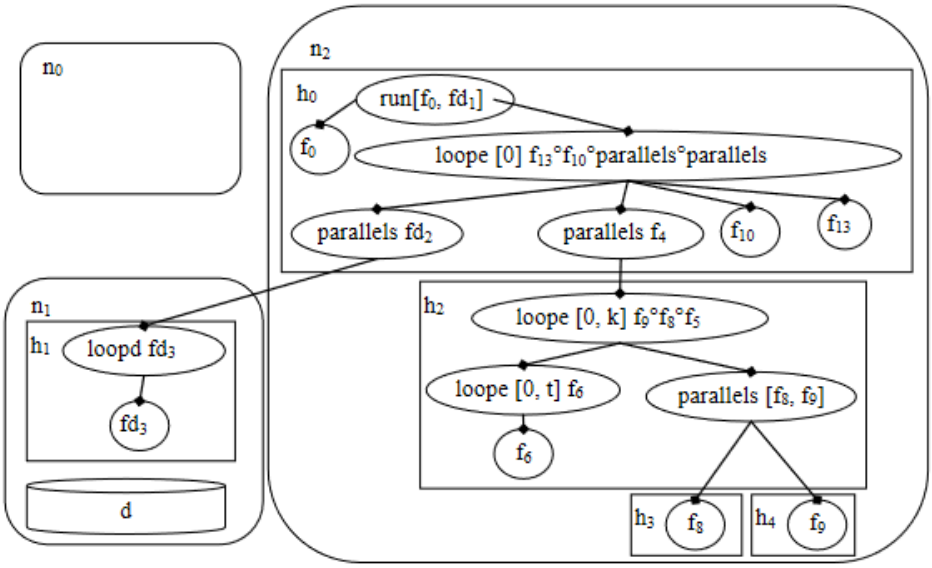

Рисунок 3.7 – Початкова розміщення виконавців для паралельного алгоритму 1R

Необхідно зауважити, що вузол n_0 залишився які не зайняті, тому що розміщення на ньому виконавців призвело б до збільшення мережевого трафіку і часу виконання алгоритму аналізу даних. На етапі виконання алгоритму аналізу даних 1R при запуску виконавця h_1 необхідно виконати тиражування, тому що він виконує розпаралелювання за даними. Так як ФОМ можуть виконуватися в з використанням загальної пам'яті, то їх слід розмістити на одному вузлі, виконавши розпаралелювання за кількістю доступних ниток.

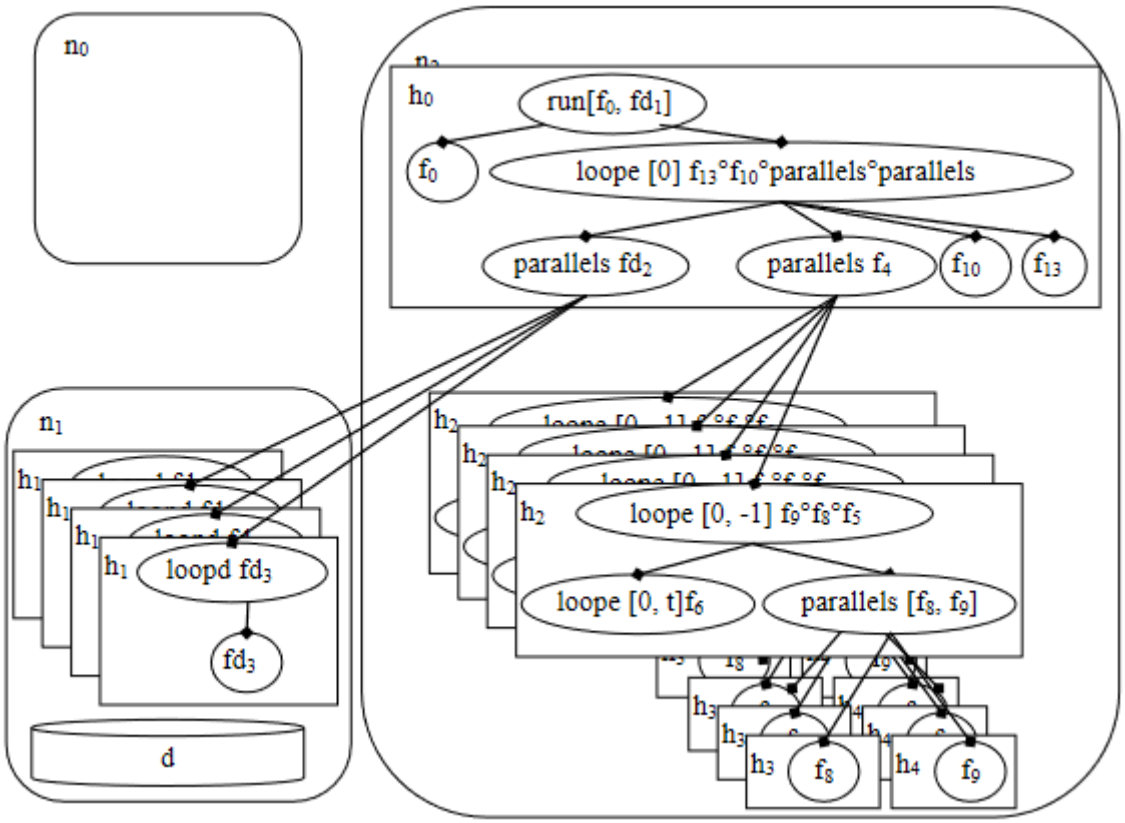

Рисунок 3.8 – Розміщення виконавців для паралельного алгоритму 1R на етапі виконання

Концепція аналізу даних передбачає наявність основних принципів вилучення знань, можливо не оформлених у вигляді алгоритму (тобто відсутня послідовність застосування даних принципів для побудови моделі знань). Такі принципи можуть бути виражені у вигляді математичних функцій, правил або методів аналізу окремих елементів набору даних (векторів, атрибутів, значень і т.п.). Дані принципи можуть бути раніше відомі і використовуватися в існуючих алгоритмах або бути новими.

Основними етапами побудови паралельного алгоритму аналізу розподілених даних є:

- побудова моделі знань у вигляді масиву елементів (2.6) для алгоритму аналізу;

- формування композиції функції f0 типу і ФОМ fi, i = 1..n у вигляді f = fn°fn-1°...°fs °...°fr °...°f1°f0;

- розпаралелювання алгоритму аналізу даних представленого у вигляді композиції функцій відповідно до виду зберігання даних;

- розміщення алгоритму аналізу даних в заданому середовищі виконання.

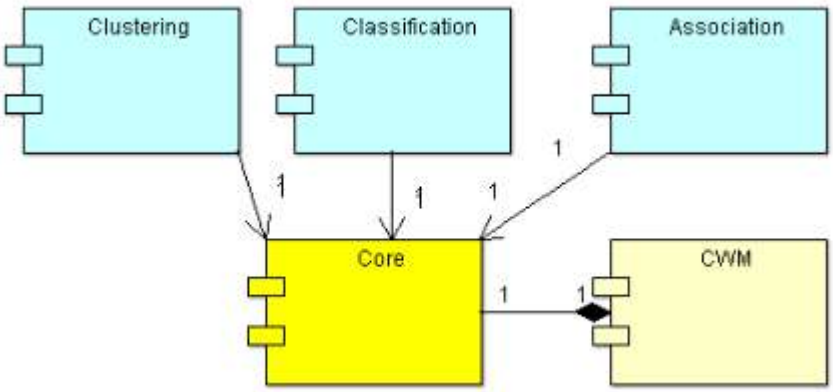

Рисунок 3.8 – Початкова розміщення виконавців для паралельного алгоритму 1R

Бібліотека DXelopes містить необхідні базові класи для побудови паралельних алгоритмів аналізу даних. Вона розділена на наступні модулі: - ядро бібліотеки (Core) побудоване відповідно до стандартом CWM і включає в себе:

- засоби роботи з даними, запозичені з бібліотеки Xelopes;

- засоби збереження результатів, відповідно до запропонованим поданням моделі знань;

- засоби для настройки завдань і алгоритмів аналізу даних, запозичені з бібліотеки Xelopes;

- базові класи для реалізації ФОМ, відповідно до функціональною моделлю алгоритму аналізу даних;

- програмну реалізацію виконавців для виконання в паралельній і розподіленої середовищі;

- алгоритми класифікації (Classification);

- алгоритми кластеризації (Clustering);

- алгоритми пошуку асоціативних правил (Association). Останні три модуля використовують ядро і не залежать один від одного.

Для реалізації підходу побудови алгоритмів аналізу даних з набору ФОМ в бібліотеку були додані відповідні класи. Вони дозволяють реалізувати окремі блоки алгоритму як ФОМ, інтегрувати їх в алгоритм і виконати його. У бібліотеці ФОМ описується класом Step (рисунок 3.9).

Реалізація блоку алгоритму міститься в методі execute, який викликається з публічного методу runStep. Виклик методу execute призводить до виконання блоку алгоритму. Даний метод відповідно до вираження (2.15), в якості вхідних аргументів приймає вхідний набір даних inputData і модель знань model. Як результат він повертає модель знань.

Всі функціональні блоки, з яких будуються алгоритми аналізу даних, повинні успадковуватися від класу Step і реалізовувати метод execute. Таким чином, всі ФОМ матимуть уніфікований інтерфейс. Для задоволення вимогу чистоти функції метод execute в класі Step повинен працювати тільки з переданими йому аргументами (вхідний набір даних inputData і модель знань model).

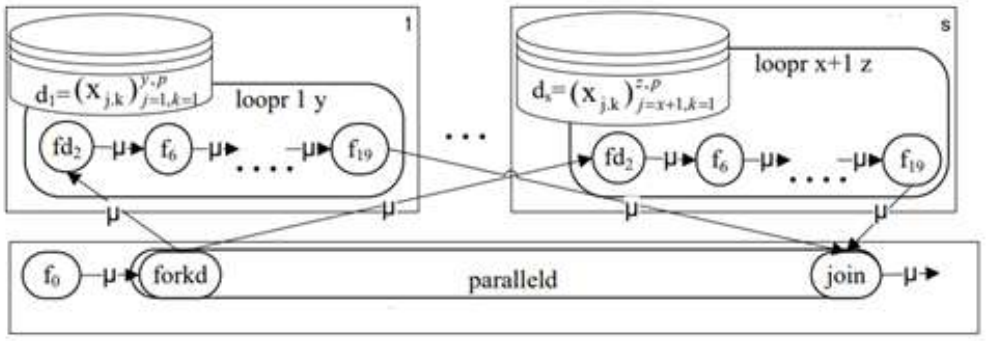

Рисунок 3.9 – Оптимізація існуючого алгоритму

Для реалізації композиції ФОМ використовується клас StepSequence. Він реалізує функцію run (2.16). Так як функція run також є ФОМ, то клас

StepSequence є спадкоємцем класу Step. Об'єкт такого класу містить поле sequence - набір об'єктів класу Step, відповідних ФОМ алгоритму, що виконуються послідовно. Результат виконання кожного ФОМ з композиції, що повертається методом execute, передається в якості аргументу наступного ФОМ в метод execute.

Оператор умовного виконання, відповідний функції dec (2.17), реалізується класом DecisionStep. Відповідно до функціональної моделлю умовний перехід є ФОМ, отже, клас DecisionStep також успадковуються від класу Step.

Метод execute класу DecisionStep реалізує умовний перехід. Для цього в класі визначено абстрактний метод condition, відповідний функції c. Для реалізації ФОМ ft і ff в класі реалізовані відповідно два поля:

- trueBranch, який реалізує композицію ФОМ ft, що виконуються при виконанні умови c;

- falseBranch, який реалізує композицію ФОМ ff, що виконуються при невиконанні умови c.

Таким чином, метод execute перевіряє умова переходу, викликаючи метод condition, реалізований в класі спадкоємця, і в залежності від результату викликає виконання однієї з послідовностей. Клас CycleStep реалізує функцію loop (2.20). Вона також є ФОМ, отже, також є спадкоємцем класу Step. M

Метод execute класу CycleStep реалізує циклічний виклик композиції функціональних блоків ft, заданих полем iteration. Цикл виконується до тих пір, поки вірно умова, що обчислюється методом initLoop (), відповідним функції c. Для ініціалізації циклу використовується метод initLoop, відповідний функції finit. Дії, що виконуються після кожної операцією і відповідні функції fafter, реалізуються шляхом afterIteration.

Для алгоритмів DataMining характерним є обробка даних по векторах (2.21), а також по атрибутам (2.22). Для їх реалізації додані класи VectorsCycleStep і AttributesCycleStep. Обидва вони є спадкоємцями класу

CycleStep і реалізують його методи, описані вище. Обробка кожного вектора або атрибута визначена композицією ФОМ - iteration. Для інтеграції ФОМ в алгоритм аналізу даних визначено клас MiningAlgorithm. Він містить в собі композицію всіх ФОМ, складових алгоритм аналізу даних - steps. Конструювання алгоритму з ФОМ виконується в методі initSteps ().

Побудова моделі знань здійснюється методом buildModel. Він викликає метод runAlgorithm, який в свою чергу викликає метод execute послідовності steps.

3.2 Класи для паралельного виконання алгорітмов аналізу даних

Основною перевагою декомпозиції алгоритмів аналізу даних на ФОМ є можливість перетворення його до паралельної формі. В бібліотеці для цього доданий клас ParallelStep (рисунок 3.10), який реалізує функцію parallel (3.1). Будучи ФОМ, він також успадковується від класу Step. Його спадкоємці визначають конкретний спосіб розпаралелювання: за даними або за завданнями.

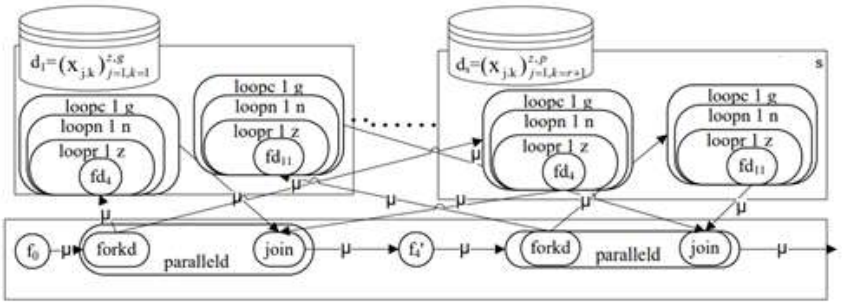

Рисунок 3.10 – Діаграма класів для реалізації паралельної форми алгоритмів аналізу даних

Для паралелізації за даними використовується клас ParallelByData. Він включає в себе послідовність ФОМ - branch, яка виконується паралельно. У

разі паралелізації за даними, послідовність є загальною для всіх потоків і застосовується до різних частин оброблюваних даних. Для паралелізації за завданнями використовується клас ParallelByTask. Він включає в себе список branches різних послідовностей ФОМ. Кожна з цих послідовностей виконується паралельно і може обробляти одні й ті ж дані.

3.3 Середовище паралельного і розподіленого виконання алгоритмів аналізу даних

Для реалізації середовища паралельного і розподіленого виконання алгоритмів аналізу даних, в бібліотеку додані відповідні класи. Середовище виконання описується абстрактним класом ExecutionEnvironment (Рисунок 3.11).

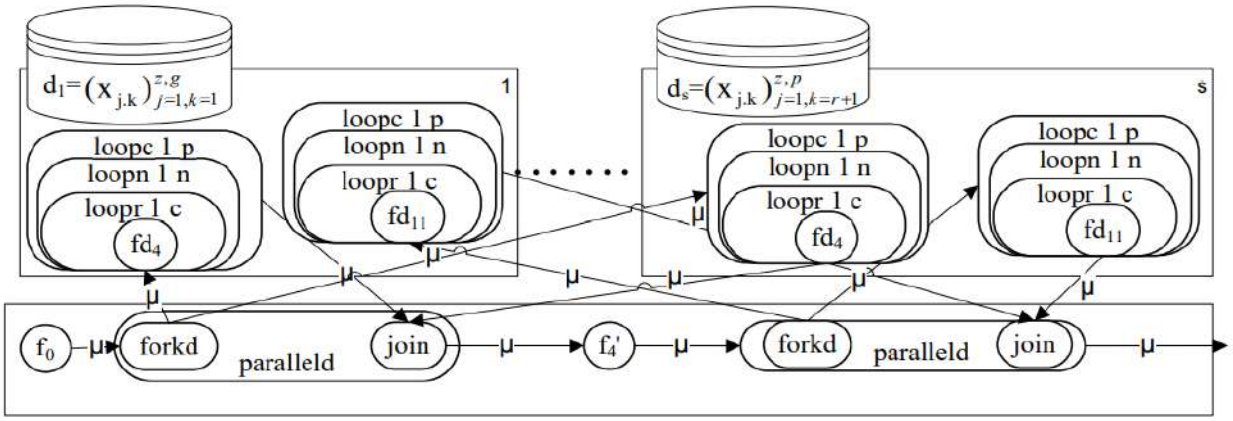

Рисунок 3.11 – Діаграма класів середовища виконання алгоритмів аналізу даних

Вона включає в себе виконавців, реалізованих як об'єкти класу ExecutionHandler, безпосередньо, виконують ФОМ. Клас ExecutionEnvironment має основного виконавця mainHandler h0, який виконує основну послідовність ФОМ задану в послідовності steps класу MiningAlgorithm. Крім того, середовище виконання, містить набір виконавців

mainHandler, відповідний доступних засобів паралельної або розподіленої середовища (наприклад, потокам, акторам і т.п.).

Виконавці створюються за допомогою класу ExecutionHandlerFactory. Для створення виконавців, Виконавча має примірник такої фабрики - executionHandlerFactory. З її допомогою створюється необхідна кількість виконавців, заданий як і інші характеристики середовища виконання, в настройках settings. Вони описуються класом ExecutionSettings, який дозволяє визначити наступні характеристики середовища:

- numberHandlers - кількість обробників;

- systemType - тип системи (многопоточность, модель акторів або ін.);

- memoryType

- тип пам'яті: розподілена (MemoryType.distributed) або розділена (MemoryType.shared).

Для розміщення алгоритму в середовищі виконання використовується метод deployAlgorithm. Він реалізує метод розміщення виконавців в середовищі виконання. Перераховані класи є базовими для опису конкретної середовища виконання. Спадкоємці цих класів є адаптерами між бібліотекою і конкретним засобом виконання паралельних або розподілених обчислень.

3.4 Виконання алгоритмів в паралельномму середовищі

Для виконання алгоритму аналізу даних в заданому середовищі виконання, в бібліотеці реалізований клас EMiningBuildTask, що є спадкоємцем класу MiningBuildTask стандарту CWM. Його основним призначенням є конфігурація завдання побудови моделі знань і виконання цього завдання. Конфігурація завдання полягає в установці таких сутностей даного завдання (рисунок 5.6):

- набір даних, що аналізуються, пов'язує завдання і спадкоємця класу MmmgInputStream, що забезпечує доступ до даних;

- настройки як самого завдання (об'єкт класу MiningFunctionSettings)

так і налаштувань конкретного алгоритму (об'єкт класу MmmgAlgorithmSettmgs);

- алгоритм аналізу даних (об'єкт спадкоємця класу MmmgAlgorithm), за допомогою якого буде виконуватися побудовою моделі знань аналізу даних;

- виконавча (об'єкт спадкоємця класу ExecutionEnvironment), в якій буде виконуватися алгоритм.

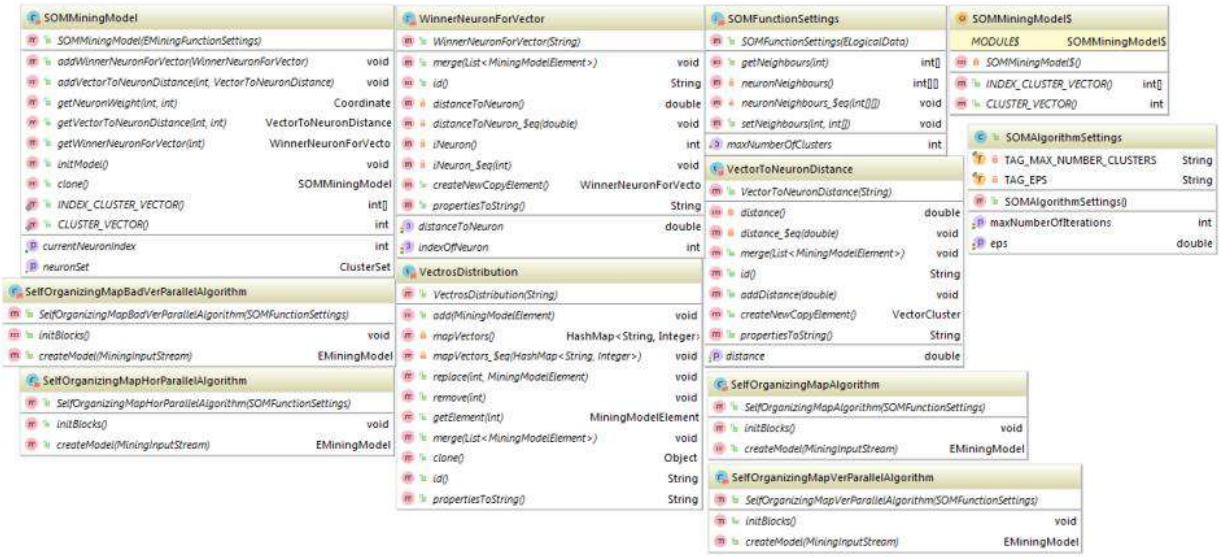

Рисунок 3.12 – Базові класи конфігурації завдання виконання алгоритму аналізу даних

Побудова моделі знань виконується викликом методу execute класу EMiningBuildTask. Налаштування виконання алгоритму визначають, в тому числі і стратегії паралельного виконання алгоритму:

- dataProcessingStrategy стратегія обробки набору даних:

- SingleDataSet обробка всього набору даних кожним паралельним виконавцем;

- SeparatedDataSet поділ даних на частини і обробка кожної частини набору даних окремим виконавцем;

- modelProcessingStrategy - вибір стратегії обробки моделі знань;

- SingleMiningModel обробка всієї моделі знань кожним паралельним виконавцем;

- SeparatedMiningModel поділ моделі знань на частини і обробка кожної частини окремим виконавцем. Для реалізації різних стратегій обробки даних використовуються спадкоємці класу MiningInputStream (рисунок 5.7) для різних форматів зберігання даних (наприклад, клас MmmgArffStream спадкоємець MiningInputStream використовується для обробки даних, що зберігаються в файлі формату ARFF).

Доступ до даних, через об'єкти даного класу, з боку паралельних виконавців залежить від властивостей середовища виконання і обраної стратегії:

- при виборі стратегії обробки всього набору даних кожним виконавцем в середовищі виконання з використанням загальної пам'яті (MemoryType.shared) іполнітель передається посилання на єдиний об'єкт спадкоємця класу MmmgArffStream;

- при виборі стратегії обробки всього набору даних кожним виконавцем в середовищі виконання з використанням розподіленої пам'яті (MemoryType.distributed) виконавцям передаються копії об'єкта спадкоємця класу MmmgArffStream, які створюються методом clone даного класу.

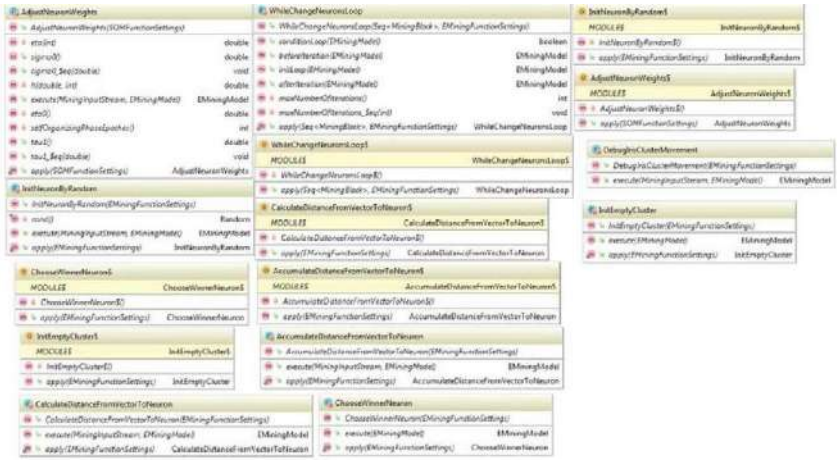

Рисунок 3.13 – Базові класи для реалізації стратегій роботи з даними аналізу даних

3.5 Аналіз результатів

Залежно від початкового вигляду опису реалізованого алгоритму аналізу даних, можливо три основні підходи додавання нового алгоритму в бібліотеку DXelopes:

- при наявності реалізації алгоритму аналізу даних, можлива її адаптація і подальша декомпозиція на ФОМ;

- при наявності алгоритму в описовому вигляді можливі два підходи:

- реалізація алгоритму у вигляді ФОМ (якщо опис алгоритму досить формалізовано і дозволяє виділити окремі блоки алгоритму);

- реалізація алгоритму в монолітному вигляді і його подальша декомпозіція на ФОМ (в разі якщо опис алгоритму недостатньо формалізована і не дозволяє декомпозировать його на блоки відразу).

Розглянемо програмну реалізацію алгоритму аналізу даних на прикладі алгоритму 1R (3.4). На першому кроці виконується програмна реалізація алгоритму до бібліотеки DXelopes. Алгоритм НЕ декомпозіруется. Основним завданням є підключення його до даних і налагодження для побудови коректної моделі знань. Програмна реалізація алгоритму 1R приведена в Додатку Б. Далі на основі виразу (2.24) реалізуємо кожен ФОМ, на які він декомпозирован, класом - спадкоємцем класу Step (Малюнок 5.14):

- цикл по атрибутам fd1 - клас AttributesCycleStep;
- цикл по векторах fd2 - клас VectorsCycleStep;
- цикл за значеннями незалежної атрибута f4 - клас CurrentAttributeValuesCycleStep;
- цикл за значеннями залежної атрибута f5 - клас CycleByTargetValues;
- функція підрахунку кількості коректних векторів fd3 - клас IncrementOneRule;
- функція додавання нового правила f8 - клас AddingOneRule;
- функція вибору кращого правила f10 - клас SelectBetterScoreRule.

Порівняння змін, які повинні бути зроблені, щоб перетворити послідовну форму алгоритму аналізу даних до послідовної формі, проводилися на вихідних текстах програм реалізованих в описаній бібліотеці алгоритмів:

- побудови класифікаційних правил 1R;

- побудови ймовірнісної моделі NaiveBayes;

- кластеризації kMeans; - пошуку частих наборів Apriori.

Для всіх перерахованих алгоритмів були реалізовані ФОМ у вигляді відповідних класів і класи, що реалізують їх композицію (спадкоємці класу MiningAlgorithm). Така реалізація відповідає послідовній формі алгоритму. Далі, відповідно до запропонованої методики, були визначені паралельні форми алгоритму для аналізу, централізовано зберігаються даних, а також горизонтально і вертикально збережених даних. Реалізовані форми алгоритмів були перевірені на однакових тестових даних. Отримані результати збігаються, що доводить правильність роботи послідовної і паралельних реалізацій алгоритмів у вигляді композиції ФОМ.

З огляду на, що при реалізації різних варіантів самі ФОМ не змінювалися, порівняно підлягали тільки класи, що реалізують їх композиції (класи спадкоємці класу MiningAlgorithm). Для порівняння змін в коді, використовувалася метрика, що вимірює виконуються строчки коду (ELOC - Executable Lines of Code).

З огляду на те, що в класі, що реалізовує композицію ФОМ, виконувані рядки коду створюють об'єкти відповідних класів, то дана метрика є адекватною.

Результати експериментів показують невеликий відсоток (від 1% до 10%) змін, що вносяться до програмну реалізацію алгоритмів аналізу даних при їх перетворенні з послідовної форми в паралельну. У разі алгоритму Naïve Bayes вищий відсоток (19%) пояснюється загальним невеликим числом рядків коду. В абсолютному вираженні при розпаралелювання для аналізу:

- централізовано зберігаються даних число змінюваних рядків (6

ELOC) визначається числом місць розпаралелювання (колонка 5);

- розподілених даних більшість змін (до 8 ELOC) пов'язано з оптимізацією циклів і як наслідок безліччю перестановок в композиції ФОМ (колонки 6 і 7).

Необхідно зауважити, що в обох випадках при перетворенні не змінюються програмні реалізації самих ФОМ. При цьому саме ця частина алгоритмів аналізу даних є найбільш ємною (становить від всього алгоритму від 70 до 90%). Таким чином, можна зробити висновок, що за рахунок локалізації змін у композиції ФОМ, перетворення послідовної форми алгоритму аналізу даних в паралельну, з урахуванням виду зберігання даних, вимагає зміни не більше 10% програмного коду.

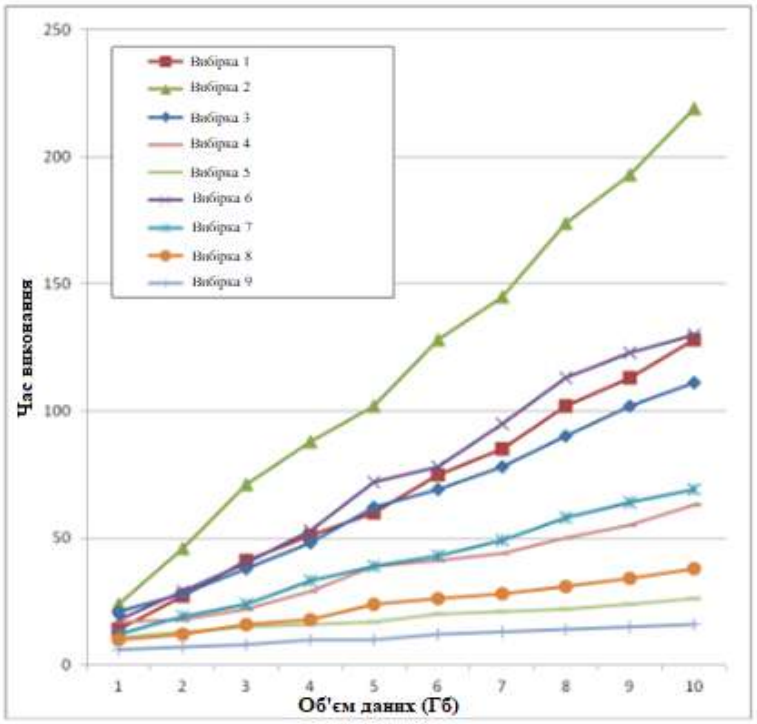

Рисунок 3.14 – Порівняння часу виконання паралельного алгоритму 1R паралельного алгоритму Native Bayes

Також необхідно зауважити, що ефективність розпаралелювання алгоритмів аналізу даних зростає зі збільшенням обсягу аналізованих даних (

рисунок 3.14). Так при обсязі даних в 1 Гб ефективність для 4-12 ядер практично не змінюється. У той час як для даних обсягом понад 2 Гб вона продовжує збільшуватися. Це пояснюється відповідністю часу, витраченого на виконання аналізу даних і часу на розпаралелювання (пов'язаного зі створенням / завершенням потоків, запуском потоків, поділом циклів і т.п.).

При цьому час, необхідний для розпаралелювання, збільшується з ростом числа потоків. При невеликих обсягах даних час аналізу можна порівняти з часом розпаралелювання для 4 і більше потоків. При збільшенні обсягу даних час аналізу зростає, при цьому час розпаралелювання не змінюється.

Таким чином, ефективність розпаралелювання найбільш ефективна при аналізі великих обсягів даних. Апроксимуємо отримані залежності ефективності від обсягу даних, можна зробити висновок, що зі збільшенням обсягів даних, масштабованість розпаралелювання буде збільшуватися. Реалізація всіх варіантів розпаралелювання алгоритмів аналізу даних, в бібліотеці DXelopes показує кращий результат, ніж їх реалізація в Apache Spark MLLib.

Запропонований підхід має кращий результат, як при збільшенні обсягу аналізованих даних, так і при і збільшенні числа ядер. Це пояснюється використанням в Apache Spark розподіленої пам'яті і необхідністю подальшого об'єднання отриманих результатів, що вимагає додаткового часу. Висока ефективність алгоритму Naïve Bayes пояснюється його особливостями. Він виконує один прохід за даними і формує модель знань, яка не залежить від обсягу даних (залежить тільки від кількості атрибутів). Крім того, він дозволяє побудова моделі знань в загальній пам'яті, тому що не має жорстких інформаційних залежностей між блоками алгоритму.

Це дозволяє виконувати алгоритм без додаткових операцій по об'єднанню результатів, отриманих паралельними гілками алгоритму. Очевидно, що в алгоритмах з більш складною структурою не вдасться досягти такої ж ефективності.

Так алгоритм Apriori, має гіршу ефективність. Особливо низькими результати виявилися для Apache Spark MLLib. Це пов'язано з великим розміром моделі знань, що формується алгоритмом Apriori. Число елементів такої моделі знань безпосередньо залежить від обсягу аналізованих транзакцій і входять до неї об'єктів. При цьому, через необхідність об'єднання моделей знань, побудованих в розподіленої пам'яті Apache Spark, час виконання значно збільшується. Таким чином, декомпозиція послідовного алгоритму на ФОМ уповільнює виконання алгоритму.

Однак, просте перетворення їх до паралельної формі, дозволяє значно прискорити виконання алгоритму на багатоядерному процесорі. При цьому ефективність підвищується зі збільшенням обсягу аналізованих даних і при невеликих розмірах споруджуваної моделі знань.

ВИСНОВКИ

Проведено аналіз існуючих моделей, методів і засобів побудови паралельних алгоритмів, який показав, що вони не дозволяють виконувати побудову паралельних алгоритмів аналізу даних з урахуванням умов їх виконання. Представлена модель знань, яка описує знайдені в даних закономірності у вигляді лісу дерев уніфікованих елементів і визначає функції, що забезпечують їх паралельне побудова в загальній і розподіленої пам'яті. Модифікований метод паралельного алгоритму аналізу даних, що дозволяє проводити розпаралелювання, як за даними, так і за завданнями, для виконання в загальній і розподіленої пам'яті. Розроблено бібліотека паралельних алгоритмів аналізу даних для виконання в заданому середовищі, що включає в себе програмні реалізації запропонованих моделей і методів. Проведено експериментальні дослідження побудови паралельних алгоритмів аналізу даних з урахуванням умов виконання, які підтвердили достовірність отриманих результатів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Варшавский П.Р., Еремеев А.П. Моделирование рассуждений на основе прецедентов в интеллектуальных системах поддержки принятия решений // Искусственный интеллект и принятие решений. 2009. № 2. С. 45-47.

2. Варшавский П.Р., Еремеев А.П. Методы правдоподобных рассуждений на основе аналогий и прецедентов для интеллектуальных систем поддержки принятия решений// Новости искусственного интеллекта. - 2006. - №3. С. 39-62.

3. Финн В.К. Об интеллектуальном анализе данных // Новости искусственного интеллекта, №3, 2004, С. 3-19.

4. W. Frawley, G. Piatetsky-Shapiro, C. Matheus Knowledge Discovery in Databases: An Overview. - AI Magazine. - 1992. pp. 213-228.

5. Kitchin Rob. The Data Revolution. United States: Sage. 2014, p. 6.

6. Piatetsky-Shapiro G, Frawley W J. Knowledge Discovery in Databases. USA: MIT Press, 1991.

7. Agrawal R., Mannila H., Srikant R., Toivonen H. and Verkamo I. Fast Discovery of Association Rules. In Advances in Knowledge Discovery and Data Mining, eds. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Menlo Park, Calif.: AAAI Press, 1996, pp. 307-328.

8. Fayyad U., Piatetsky-Shapiro G., Smyth P., Advances in Knowledge Discovery and Data Mining, (Chapter 1), AAAI/MIT Press, 1996.

9. Барсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP. // 2-еизд., - СПб: БХВ-Петербург, 2007.

10. А.А. Барсегян, И.И. Холод, М.Д. Тесс, М.С. Куприянов, С.И. Елизаров. Анализ данных и процессов. 3-е изд. – СПб.: БХВ-Петербург, 2009.

11. Интеллектуальный анализ данных средствами MS SQL Server 2008 - [Электронный ресурс].

12. Data Mining – технология добычи данных – [Электронный ресурс]. URL: http://bourabai.ru/tpoi/datamining.htm: (дата обращения: 05.01.2017).

13. Дюк В.А., Самойленко А.П. Data Mining: учебный курс СПб.: Питер, 2001.

14. Чубукова И.А. Data Mining, БИНОМ. Лаборатория знаний, Интернет- университет информационных технологий - ИНТУИТ.ру, 2006.

15. Филипов В.А. Интеллектуальный анализ данных: методы и средства. М.: Эдиториал УРСС, 2001.

16. Дюран Б., Оделл П. Кластерный анализ. М.: Статистика, 1977.

17. А.Н.Тихонов, В.Я.Арсенин. Методы решения некорректных задач. Наука, Москва, 1974.

18. Judea Pearl, Stuart Russell. Bayesian Networks. UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.

19. Ферстер Э., Ренц Б. Методы корреляционного и регрессионного анализа = Methoden der Korrelation - und Regressiolynsanalyse. – М.: Финансы и статистика, 1981.

20. Управленческий учет: учебник / под ред. А.Д. Шеремета. 4-е изд. – М.: ИНФРА-М, 2009.

21. Дяченко В.О., Козлов Ю.В., Синяокий А.О. // Аналіз ефективності навчання штучних нейронниї мереж. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей дванадцятої міжнародної науково-технічної конференції 27-28 квітня 2022 р., т.1. Баку-Харків-Жиліна. 2022 р. С.74.