DOI: https://doi.org/10.26642/ten-2026-1(97)-264-270
УДК 04.021:004.75

Я.С. Садовий, аспірант
В.В. Воротніков, д.т.н., проф.
Державний університет «Житомирська політехніка»

RD-GST: онлайн метод побудови розподілених узагальнених суфіксних дерев

Узагальнені суфіксні дерева (УСД) є фундаментальними структурами даних для точного пошуку підрядків із лінійною часовою складністю $O(m)$ відносно довжини запиту m, незалежно від розміру індексованого корпусу. Завдяки цій властивості УСД широко застосовуються в аналізі геномних послідовностей та індексуванні великих текстових масивів. Попри це, переважна більшість розподілених реалізацій УСД є офлайн-орієнтованими: вони будують індекс для статичного набору рядків, а надходження нових даних вимагає повної або істотної реконструкції структури. Це унеможливлює їх застосування в аналітичних платформах із безперервними потоками даних, що потребують одночасно високої швидкості та точності пошуку.

Метою роботи є розроблення методу побудови розподіленого узагальненого суфіксного дерева, придатного для інкрементальної обробки потокових даних у кластерному середовищі без повної перебудови індексу при надходженні нових рядків.

Запропоновано метод RD-GST, який поєднує інкрементальну стратегію конструювання суфіксного дерева з хеш-орієнтованою схемою розподілу рядків між вузлами обчислювального кластера. Метод реалізовано в архітектурі «координатор – робочі вузли» у контейнеризованому середовищі на базі Docker та AWS ECS. Принцип локальності оброблення мінімізує обсяг міжвузлових комунікацій і забезпечує інкрементальне оновлення індексу без повної реконструкції.

Для оцінювання ефективності проведено порівняльний експеримент із методом DGST на підвибірках корпусу DNS-імен обсягом 100–600 млн символів за однакових обчислювальних ресурсів. RD-GST забезпечує прискорення у 3,2–5,1 разу та пропускну здатність 5,1–5,5 млн символів/с проти 0,9–1,7 млн символів/с для DGST. Перевага у швидкодії супроводжується інтенсивнішим споживанням оперативної пам'яті, що потребує пропорційного нарощування ресурсів зі збільшенням обсягу корпусу.

Доведено практичну ефективність онлайн-підходу для потокової обробки текстових корпусів у реальному часі. Усі результати відтворювані на основі відкритих програмних реалізацій за узгоджених експериментальних умов.

Ключові слова: *узагальнене суфіксне дерево; розподілені обчислення; потокові дані; структури даних; текстовий пошук.*

Актуальність теми. Суфіксні дерева (СД) належать до фундаментальних структур даних для розв'язання задачі точного пошуку підрядка із часовою складністю $O(m)$ відносно довжини запиту, що зумовлює їхнє широке застосування як базового інструменту індексування великих текстових корпусів [1; 2]. Попри значний прогрес у розробленні серійних і паралельних алгоритмів побудови, зокрема підходів на основі зовнішньої пам'яті та розподілених обчислень [3; 4], переважна більшість наявних практичних рішень залишається офлайн-орієнтованою: індекс формується для статичного набору рядків, тоді як надходження нових даних зумовлює необхідність повної або значної реконструкції структури.

Для сучасних аналітичних платформ, що функціонують в умовах безперервних потоків різнорідних даних (зокрема, DNS-імен, мережевих журналів, логів застосунків), визначальними вимогами постають можливість інкрементального оновлення, стійке горизонтальне масштабування та мінімізація міжвузлових комунікацій. Офлайн-орієнтовані підходи, такі як DGST [4], забезпечують високу пропускну здатність на значних обсягах даних, проте не дають змоги ефективно оновлювати індекс у режимі реального часу без перерахунку істотних фрагментів структури. Водночас попередні результати автора щодо застосування префіксної сегментації для розподіленої побудови узагальнених суфіксних дерев продемонстрували потенціал детермінованого розподілу в контексті зниження комунікаційних витрат, однак потребують подальшого узагальнення на випадок менш передбачуваних вхідних потоків [5].

Аналіз останніх досліджень та публікацій, на які спираються автори. Початкові ефективні підходи до побудови суфіксних дерев були орієнтовані на виконання в межах одного обчислювального вузла з використанням оперативної пам'яті. Алгоритм МакКрейта [2] забезпечує побудову за лінійний час у середньому випадку, спираючись на поступове розширення активних шляхів. Подальша робота Укконена [1] запропонувала онлайновий (у сенсі покрокового оброблення символів) лінійний алгоритм, що уможливлює додавання суфіксів без перерахунку всієї структури. Хоча обидва підходи

характеризуються оптимальною асимптотичною складністю, їхня практична масштабованість обмежена ємністю пам'яті одного вузла, а інкрементальність в умовах розподіленого оброблення не забезпечується.

Для подолання обмежень оперативної пам'яті було запропоновано рішення із залученням зовнішньої пам'яті та паралельних реалізацій. ERA [3] поєднує серійні та паралельні фази, ефективно використовуючи дискові структури; втім, цей підхід реалізує офлайн побудову. Альтернативним напрямом є суфіксні масиви – компактніша структура, що потребує значно менше пам'яті порівняно із суфіксними деревами. Haag та ін. [6] запропонували розподілену адаптацію алгоритму DCX побудови суфіксного масиву, що скорочує потребу в оперативній пам'яті до 14–26-кратного розміру вхідних даних та досягає прискорення до 5 разів порівняно з попередніми розподіленими алгоритмами. Проте суфіксні масиви, на відміну від суфіксних дерев, не підтримують додавання нових рядків до індексу після його побудови, що ускладнює їхнє безпосереднє застосування в сценаріях безперервних потоків даних.

З появою розподілених платформ оброблення даних було розроблено алгоритми побудови СД/УСД, здатні масштабуватися на кластерах, проте більшість із них також залишаються пакетно-орієнтованими. DGST [4] демонструє ефективну та масштабовану побудову узагальнених суфіксних дерев на розподілених платформах із паралелізмом за даними (зокрема, на основі середовища Apache Spark). В архітектурному відношенні DGST використовує фазове компонування та інтенсивний обмін даними між виконавцями під час конструювання структур. Попри належну масштабованість і продуктивність на значних обсягах, DGST є офлайн методом: поповнення корпусу новими рядками у типовому сценарії потребує повторення ключових етапів побудови. Глибовець та Діденко [7] здійснили систематизацію підходів до побудови узагальнених суфіксних дерев на розподілених платформах, підтвердивши, що наявні рішення залишаються переважно пакетними та не забезпечують підтримки інкрементальних оновлень.

Проблему повноцінної онлайн побудови суфіксних дерев для колекції рядків досліджували Takagi та ін. [8], які запропонували алгоритм із часовою складністю $O(N \log \sigma)$ та лінійними за пам'яттю вимогами, де новий символ може бути доданий до довільного тексту колекції у будь-який момент часу. Цей повністю онлайновий підхід є природною теоретичною передумовою для перенесення принципу інкрементальності в розподілене середовище, що становить предмет цієї роботи. Суміжним напрямом досліджень є онлайн-алгоритми побудови суфіксних структур без зберігання вхідного тексту в пам'яті [9]: їхня лінійна часова складність за посимвольної обробки підтверджує доцільність аналогічного підходу в розподіленому контексті.

Ідеї Укконена [1] природно зумовлюють інкрементальність побудови, проте їхнє перенесення у багатовузлове середовище вимагає детермінованого розподілу даних та мінімізації обсягу міжвузлових комунікацій. У попередній роботі автора досліджено сегментацію за префіксами [5] для побудови розподіленого УСД та показано, що обґрунтований розподіл знижує комунікаційні витрати й стабілізує навантаження. Водночас префіксна сегментація може виявлятися чутливою до розподілу частот символів та специфіки предметної області корпусу, що обмежує її універсальність.

Метою статті є розроблення та експериментальне оцінювання онлайн методу побудови розподіленого узагальненого суфіксного дерева (RD-GST), що забезпечує інкрементальне оновлення індексу та стійке горизонтальне масштабування у хмарному середовищі. У межах поставленої мети досліджується перевага у часі побудови порівняно з офлайн методом DGST за вирівняних обчислювальних ресурсів, аналізується стабільність масштабування RD-GST зі зростанням обсягу даних за незмінної конфігурації кластера, а також оцінюється вплив хеш-орієнтованого розподілу на комунікаційні витрати та локальність оновлень у багатовузловому середовищі.

Викладення основного матеріалу. Метод RD-GST реалізовано у контейнеризованому середовищі відповідно до архітектурної моделі «координатор – робочі вузли». Координатор приймає потік вхідних рядків, обчислює значення хеш-функції розподілу, здійснює маршрутизацію рядка на відповідний вузол, присвоює глобальні індекси входжень, відстежує метрики продуктивності та ініціює формування контрольних точок. Робочий вузол підтримує локальний фрагмент узагальненого суфіксного дерева та виконує інкрементальне оновлення структури за принципом побудови Укконена [1]. Компоненти системи розгортаються у середовищі Docker та оркеструються на платформі AWS ECS із використанням сервісу виявлення Consul; подібна модель «координатор – робочі вузли» є усталеним підходом у сучасній мікросервісній оркестрації хмарних кластерів [10], де координатор відповідає за балансування навантаження та маршрутизацію запитів між однорідними робочими вузлами.

Архітектуру методу RD-GST схематично зображено на рисунку 1. Потік вхідних рядків надходить до координатора, який обчислює значення хеш-функції та маршрутизує кожен рядок на відповідний робочий вузол. Після первинного призначення рядок обробляється виключно на визначеному вузлі: символи не переміщуються між вузлами, а координатор не бере участі в локальній побудові дерева. Такий підхід забезпечує однонаправленість потоку даних і усуває необхідність міжвузлової синхронізації на кожному кроці вставки. Зворотний канал від робочих вузлів до координатора використовується виключно для передавання метрик продуктивності.

Обробка пошукових запитів здійснюється через той самий координатор. На відміну від префіксних схем розподілу, у яких запит може бути структурно маршрутизований на підмножину вузлів, хеш-орієнтований розподіл не зберігає лексичної суміжності рядків. Тому в загальному випадку пошук підрядка виконується паралельно на всіх робочих вузлах, а координатор об'єднує отримані результати перед поверненням відповіді. Часова складність пошуку залишається $O(m + occ)$, де m – довжина шаблону, occ – кількість входжень, і не залежить від розміру корпусу. У випадку точного пошуку рядка Q (exact match) координатор може застосувати ту саму хеш-функцію $H(Q)\ mod\ N$ для визначення вузла, який містить відповідний рядок, і адресувати запит лише до нього. Це дозволяє уникнути звернення до всієї множини робочих вузлів та суттєво знизити комунікаційні витрати в сценаріях, де структура запиту збігається з одиницею розподілу.

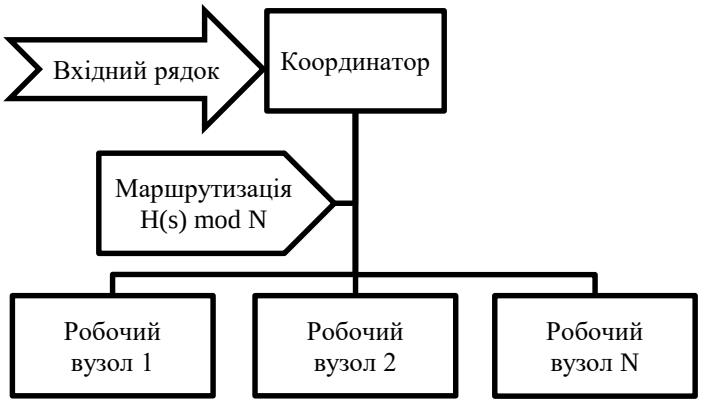

Рис. 1. Архітектура методу RD-GST: схема додавання рядка

Основу локальної побудови на кожному робочому вузлі становить алгоритм побудови узагальненого суфіксного дерева для колекції текстів $\mathcal{T} = \{T_1, T_2, \dots, T_K\}$, що поповнюється новими рядками. На відміну від класичного алгоритму Укконена [1], який будує суфіксне дерево для одного рядка T за час $O(n)$, де $n = |T|$, спираючись на три ключові механізми – активну точку, суфіксні посилання та листові ребра з відкритим кінцем, – у RD-GST алгоритм працює в напівонлайновому режимі: нові рядки $T_{k+1}, T_{k+2}, ..$ послідовно додаються до існуючої структури, тоді як уже додані рядки $T_1, \dots, T_k$ залишаються незмінними. Такий режим відповідає типовим задачам індексування журналів, моніторингу подій та накопичення текстових даних, де нові записи безперервно надходять, але історичні дані не модифікуються.

Процес додавання нового рядка T_{k+1} відбувається таким чином. Спочатку активна точка скидається на корінь дерева, оскільки новий рядок не має спільного контексту з попередньою побудовою. Далі виконується стандартна процедура Укконена: для кожного символу $T_{k+1}[i]$ дерево розширюється відповідно до правил вставлення суфіксів. Після оброблення останнього символу додається єдиний термінатор \$, спільний для всіх рядків колекції, що гарантує завершення всіх суфіксів у листках. Розрізнення суфіксів різних рядків забезпечується розширеними мітками листків: кожен новостворений лист отримує мітку $(k + 1, j)$, де j – позиція початку суфікса у рядку T_{k+1}. Ключовою особливістю є збереження структури дерева між додаванням послідовних рядків: після завершення оброблення Tk дерево залишається у пам'яті та слугує основою для інтеграції T_{k+1}, що принципово відрізняє запропонований підхід від пакетних методів побудови.

Загальна часова складність побудови узагальненого суфіксного дерева для колекції з K рядків загальною довжиною $N = \sum_{k=1}^{K} |T_k|$ становить $O(N)$, а просторова складність – $O(N)$, оскільки кількість вузлів обмежена $2N - 1$. Пошук шаблону довжини m виконується за час $O(m + occ)$, де occ – кількість входжень, що є оптимальним і не залежить від розміру колекції. На відміну від повністю онлайнового підходу Takagi та ін. [8], де довільний рядок колекції може зростати у будь-який момент часу в межах одного вузла, RD-GST здійснює розподіл рядків між вузлами кластера, зберігаючи аналогічну інкрементальність на рівні кожного локального фрагмента дерева.

Мітки ребер зберігаються як рядкові об'єкти із застосуванням спільного словника для усунення дублікатів, що зменшує обсяг зайнятої пам'яті завдяки повторному залученню однакових міток. Колекція дочірніх вузлів реалізується адаптивною структурою: для вузлів із малою кількістю переходів використовується компактний відсортований масив, для вузлів із великою кількістю – хеш-таблиця. Це дозволяє зберігати $O(k)$ пам'яті на вузол, де k – фактична кількість дочірніх посилань, замість $O(\sigma)$ для масиву фіксованого розміру.

Нехай $H(\cdot)$ – детермінована хеш-функція з рівномірним розподілом. Для кожного рядка s координатор обчислює:

$$p = H(s) \bmod N, \quad (1)$$

де N – кількість робочих вузлів, і призначає рядок s вузлу p. Такий розподіл забезпечує рівномірність навантаження для різнорідних корпусів, гарантує детермінованість маршрутизації та мінімізує міжвузлові комунікації завдяки локальності побудови.

На відміну від префіксної сегментації [5], де розподіл є чутливим до частот символів та специфіки предметної області корпусу, хеш-орієнтований розподіл не потребує попереднього аналізу даних та виявляється стабільнішим для непередбачуваних потоків. Водночас префіксна сегментація здатна забезпечити кращу локальність для вузькоспеціалізованих корпусів, що визначає перспективу подальших досліджень щодо комбінованих стратегій розподілу.

Нехай M – кількість рядків, $|s_i|$ – довжина i-го рядка. Для N вузлів очікуваний обсяг обчислень на один вузол становить:

$$\tilde{O}\left(\sum_i |s_i|/N\right), \quad (2)$$

а міжвузлова комунікація обмежується передаванням службових метаданих (керування навантаженням, контрольні сигнали, метрики). На відміну від DGST [4], де побудова складається з кількох синхронних фаз із проміжними операціями перемішування похідних структур (частотних лічильників та LCP-масивів) між вузлами, RD-GST уникає міжвузлового обміну даними під час побудови індексу.

Для оцінювання ефективності методу використано підвибірки корпусу DNS-імен [11] обсягом від 100 млн до 600 млн символів із кроком 100 млн. Рядки нормалізовано до нижнього регістру; зайві пробіли та непечатні символи вилучено. Порядок рядків зафіксовано для забезпечення відтворюваності; усі експерименти виконувалися на ідентичних підвибірках для обох методів. Систему RD-GST розгорнуто у контейнерах Docker на платформі AWS ECS із сервісом виявлення Consul. Кластер складався з 8 вузлів, кожен з яких мав 8 віртуальних ядер і 16 ГБ оперативної пам'яті (загалом 64 ядра, 128 ГБ). Базовий код DGST з офіційного репозиторію [12] було адаптовано для AWS EMR Serverless із використанням Amazon S3 як розподіленої файлової системи; зміни в реалізації та параметри запуску наведено у репозиторію автора [13]. Конфігурацію Apache Spark визначено як 64 виконавці по 1 ядру та 2 ГБ пам'яті кожен (загалом 64 ядра, 128 ГБ). Таким чином, сукупні обчислювальні ресурси для RD-GST і DGST було вирівняно. Оцінювання здійснювалося за такими показниками: загальний час побудови (мс) – час від початку завантаження підвибірки до завершення конструювання структури; пропускна здатність (символів/с) – відношення обсягу оброблених символів до загального часу побудови; прискорення:

$$S = \frac{T_{DGST}}{T_{RD-GST}}, \quad (3)$$

де T_{DGST} і $T_{RD\text{-}GST}$ – час побудови на тій самій підвибірці.

У таблиці 1 наведено результати порівняння часу побудови та похідних метрик. Пропускну здатність обчислено як $Throughput = Size/T$.

Таблиця 1

Порівняння RD-GST і DGST на корпусі DNS-імен (100–600 млн символів)

Обсяг	DGST, мс	RD-GST, мс	Прискорення	DGST, Mchars/s	RD-GST, Mchars/s
100 млн	106 169	20 716	5,12	0,94	4,83
200 млн	159 534	39 285	4,06	1,25	5,09
300 млн	212 634	54 932	3,87	1,41	5,46
400 млн	237 630	73 522	3,23	1,68	5,44
500 млн	289 266	91 644	3,16	1,73	5,46
600 млн	355 931	109 583	3,25	1,69	5,48

Джерело: розробка автора

Аналіз отриманих результатів засвідчує, що RD-GST стабільно перевершує DGST на всіх підвибірках: прискорення S варіюється від 5,12 (100 млн символів) до 3,16–3,25 на обсягах 500–600 млн символів. Середнє значення прискорення у діапазоні 200–600 млн символів становить приблизно $S \approx 3{,}7$. RD-GST досягає пропускної здатності на рівні 5,1–5,5 Mchars/s практично на всьому діапазоні обсягів, тоді як у DGST цей показник зростає від 0,94 до 1,7 Mchars/s зі збільшенням підвибірки та залишається суттєво нижчим.

Динаміку прискорення S зображено на рисунку 2. Спадання від 5,12 до 3,16 зі зростанням обсягу пояснюється збільшенням середньої глибини обходу локального дерева при вставці нових суфіксів: чим більший накопичений фрагмент УСД на вузлі, тим більше кроків потребує інтеграція кожного наступного рядка. Стабілізація S у діапазоні 500–600 млн символів свідчить про досягнення стаціонарного режиму

роботи системи. Загальна перевага RD-GST зумовлена локальністю оброблення: хеш-орієнтований розподіл фіксує належність рядка до конкретного вузла, а інкрементальна стратегія побудови оновлює локальний фрагмент дерева без міжвузлової синхронізації на кожному кроці вставки. Натомість DGST реалізує багатофазний конвеєр, у якому між стадіями відбувається перемішування похідних структур (частотних лічильників та LCP-масивів) між вузлами, а кожна наступна стадія може розпочатися лише після повного завершення попередньої. Оскільки обидві системи використовують однакову кількість ядер і обсяг оперативної пам'яті (64 ядра, 128 ГБ), спостережувана різниця пояснюється не апаратними факторами, а відмінностями в алгоритмічній організації та комунікаційній моделі.

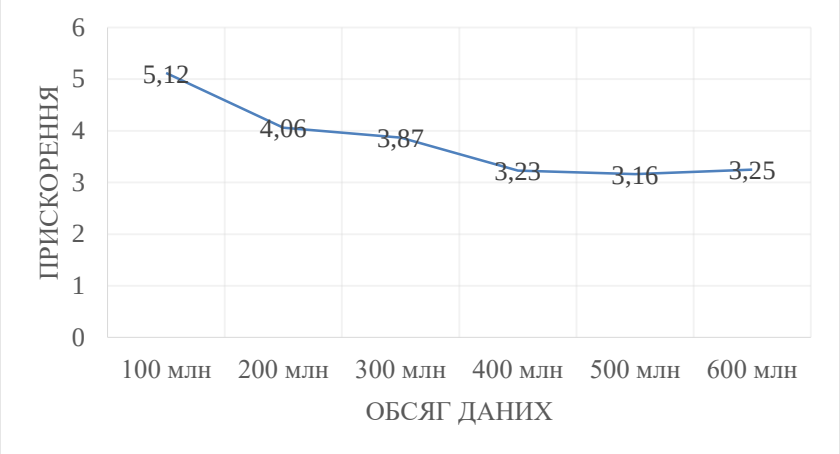

Рис. 2. Відносне прискорення розробленого методу порівняно з DGST

Аналіз використання оперативної пам'яті виявляє суттєву відмінність між підходами. У RD-GST середнє значення використання пам'яті на один робочий вузол зростає від 1,8 ГБ при 100 млн символів до 7,7 ГБ при 600 млн символів, що відповідає лінійному зростанню локального фрагмента УСД. У конфігурації DGST на платформі EMR Serverless обсяг пам'яті на виконавця обмежений виділеними 2 ГБ, що дає змогу обробляти більші корпуси за рахунок збільшення кількості виконавців без пропорційного нарощування ресурсів кожного з них. Таким чином, RD-GST забезпечує значний виграш у часі побудови за умови наявності достатнього обсягу оперативної пам'яті, тоді як DGST є ефективнішим щодо використання пам'яті у сценаріях пакетної побудови для надвеликих корпусів.

Важливою відмінністю є функціональні можливості обох підходів. DGST здійснює побудову дерева для фіксованого корпусу і не підтримує інкрементальне оновлення: надходження нових даних потребує повної реконструкції індексу. RD-GST натомість забезпечує інкрементальну побудову та підтримує виконання пошукових запитів безпосередньо під час індексації, що робить його придатним для аналітичних платформ реального часу з безперервними потоками даних. Відсутність підтримки видалення рядків із побудованого індексу є характерним обмеженням суфіксних структур і становить предмет подальших досліджень.

Слід зазначити, що наведені виміри охоплюють однотипний корпус (DNS-імена) і не враховують варіацій мережевих умов чи альтернативних конфігурацій вузлів. Верифікація методу на різнорідних даних (мережеві журнали, тексти природною мовою, аналітичні дані) та зі змінними параметрами кластерів становить предмет подальшої роботи.

Результати експериментального оцінювання вказують на практичну придатність методу RD-GST для задач, де індекс формується інкрементально, а пошукові запити надходять паралельно з оновленням корпусу. Зокрема, платформи кіберзахисту класу Threat Intelligence підтримують бази індикаторів компрометації (IoC), що містять мільярди записів — доменні імена, URL-адреси, хеші файлів та мережеві сигнатури. Такі бази поповнюються безперервно: нові шкідливі домени та сигнатури виявляються щогодини і негайно інтегруються до індексу, тоді як пошукові запити надходять паралельно — кожен мережевий запит у корпоративній інфраструктурі перевіряється проти актуальної бази в реальному часі. На відміну від хеш-таблиць, що забезпечують лише точний збіг, суфіксний індекс дозволяє виявляти шкідливі патерни як підрядки: домен evil.com буде знайдено у складі login.evil.com.ua, а характерні суфіксні структури DGA-доменів — ідентифіковано незалежно від повної форми запису. Інкрементальне оновлення індексу без його повної реконструкції є критичною вимогою в цьому контексті, оскільки затримка між виявленням загрози та її інтеграцією до індексу безпосередньо визначає ефективність захисту.

Описана модель – інкрементальне поповнення корпусу в поєднанні з інтенсивними пошуковими запитами – є загальною для широкого класу прикладних задач: індексування баз патентів і наукових публікацій, пошук гомологій у геномних базах даних, моніторинг реєстрів програмних артефактів та аналіз журналів розподілених систем. У всіх зазначених сценаріях метод RD-GST забезпечує пошук за часом $O(m + occ)$ незалежно від обсягу накопиченого корпусу, що робить його перспективним інструментом індексування у сферах, де дані накопичуються безперервно, а затримка відповіді є критичним параметром системи.

Висновки та перспективи подальших досліджень. У роботі запропоновано та експериментально досліджено метод RD-GST – онлайн підхід до побудови розподіленого узагальненого суфіксного дерева, що поєднує інкрементальну стратегію конструювання структури із хеш-орієнтованою схемою розподілу рядків між вузлами обчислювального кластера. Метод реалізовано відповідно до архітектурної моделі «координатор – робочі вузли» у контейнеризованому хмарному середовищі.

Результати експериментального порівняння з офлайн методом DGST на підвибірках корпусу DNS-імен обсягом від 100 до 600 млн символів за вирівняних обчислювальних ресурсів (64 ядра, 128 ГБ) засвідчили стабільну перевагу запропонованого методу: прискорення становить від 3,16 до 5,12 разу (середнє $S \approx 3{,}7$ у діапазоні 200–600 млн символів), а пропускна здатність сягає 5,1–5,5 Mchars/s порівняно з 0,9–1,7 Mchars/s для DGST. Спостережувана різниця пояснюється не апаратними факторами, а відмінностями в алгоритмічній організації та комунікаційній моделі: детермінований розподіл фіксує належність рядка до конкретного вузла, а інкрементальний режим побудови забезпечує оновлення локального фрагмента дерева без міжвузлової синхронізації на кожному кроці вставки. Натомість DGST реалізує багатофазний конвеєр із перемішуванням похідних структур між вузлами, де кожна наступна стадія може розпочатися лише після повного завершення попередньої.

Водночас зазначене прискорення супроводжується інтенсивнішим використанням оперативної пам’яті, що потребує пропорційного нарощування ресурсів зі збільшенням обсягу оброблюваних даних. За цим критерієм DGST виявляє вищу ефективність у сценаріях пакетної побудови для надвеликих статичних корпусів даних.

Перспективними напрямами подальших досліджень визначено: верифікацію методу на різнорідних корпусах (мережеві журнали, програмні логи, тексти природною мовою); реалізацію динамічного масштабування кластера із автоматичним перерозподілом навантаження під час роботи системи; розроблення механізмів серіалізації та відновлення стану індексу для забезпечення відмовостійкості; оптимізацію використання пам’яті шляхом застосування файлів, відображених у пам’ять (memory-mapped files), що дасть змогу знизити навантаження на оперативну пам’ять та обробляти корпуси, розмір яких перевищує її обсяг.

References:

1. Ukkonen, E. (1995), «On-line construction of suffix trees», *Algorithmica*, Vol. 14, No. 3, pp. 249–260, doi: 10.1007/BF01206331.
2. McCreight, E.M. (1976), «A space-economical suffix tree construction algorithm», *Journal of the ACM*, Vol. 23, No. 2, pp. 262–272, doi: 10.1145/321941.321946.
3. Mansour, E. et al. (2011), «ERA: efficient serial and parallel suffix tree construction for very long strings», *Proceedings of the VLDB Endowment*, Vol. 5, No. 1, pp. 49–60, doi: 10.14778/2047485.2047490.
4. Zhu, G. et al. (2019), «DGST: efficient and scalable suffix tree construction on distributed data-parallel platforms», *Parallel Computing*, Vol. 87, pp. 87–102, doi: 10.1016/j.parco.2019.06.002.
5. Sadovyi, I. and Vorotnikov, V. (2025), «The study of optimization of the segmentation process for building distributed generalized suffix trees», *Journal of Theoretical and Applied Information Technology*, Vol. 103, No. 9, [Online], available at: https://jatit.org/volumes/Vol103No9/5Vol103No9.pdf
6. Haag, M. et al. (2025), «Fast and lightweight distributed suffix array construction», *in 33rd Annual European Symposium on Algorithms (ESA 2025)*. Series. *Leibniz International Proceedings in Informatics (LIPIcs)*, Vol. 351, pp. 47:1–47:18, doi: 10.4230/LIPIcs.ESA.2025.47.
7. Hlybovets, A. and Didenko, V. (2023), «Constructing generalized suffix trees on distributed parallel platforms», *Cybernetics and Systems Analysis*, Vol. 59, No. 1, pp. 49–60, doi: 10.1007/s10559-023-00541-x.
8. Takagi, T. et al. (2020), «Fully-online suffix tree and directed acyclic word graph construction for multiple texts», *Algorithmica*, Vol. 82, doi: 10.1007/s00453-019-00646-w.
9. Hendrian, D. et al. (2024), «Linear time online algorithms for constructing linear-size suffix trie», *Theoretical Computer Science*, Vol. 1015, doi: 10.1016/j.tcs.2024.114765.
10. Marchese, A. and Tomarchio, O. (2025), «Enhancing the Kubernetes platform with a load-aware orchestration strategy», *SN Computer Science*, Vol. 6, No. 3, 224 p., doi: 10.1007/s42979-025-03712-z.
11. Domains Project, «Domains Project: processing petabytes of data», [Online], available at: https://domainsproject.org/
12. PasaLab, «DGST» [Source code], GitHub, [Online], available at: https://github.com/PasaLab/DGST
13. *GitHub: iansadovy/DGST*, [Online], available at: https://github.com/iansadovy/DGST

Садовий Ян Станіславович – аспірант кафедри інженерії програмного забезпечення Державного університету «Житомирська політехніка».

https://orcid.org/0000-0003-0387-6772.

Наукові інтереси:

- структури даних та алгоритми;
- побудова та оптимізація суфіксних дерев;
- хмарні обчислення та контейнеризація.

Воротніков Володимир Володимирович – доктор технічних наук, доцент, професор кафедри комп'ютерної інженерії та кібербезпеки Державного університету «Житомирська політехніка».

https://orcid.org/0000-0001-8584-3901.

Наукові інтереси:

- комп'ютерні мережі та мережні технології;
- мережна безпека, кібербезпека;
- керування складними інформаційними системами.

Sadovyi I.S., Vorotnikov V.V.

RD-GST: online method for constructing distributed generalized suffix trees

Generalized suffix trees (GST) are fundamental data structures for exact substring search with linear time complexity $O(m)$ with respect to query length m, regardless of the size of the indexed corpus. This property makes GSTs widely applicable in genomic sequence analysis and large-scale text corpus indexing. However, the majority of existing distributed GST implementations are offline-oriented: they construct the index for a static string collection, and the arrival of new data requires full or substantial reconstruction of the structure. This makes them unsuitable for analytical platforms operating on continuous data streams that demand both high throughput and exact search accuracy.

The objective of this work is to develop a method for constructing a distributed generalized suffix tree capable of incremental processing of streaming data in a cluster environment without full index reconstruction upon the arrival of new strings.

The proposed RD-GST method combines an incremental suffix tree construction strategy with a hash-oriented string distribution scheme across cluster nodes. The method is implemented following a coordinator–worker architecture in a containerized environment based on Docker and AWS ECS. A data locality principle minimizes inter-node communication overhead and ensures incremental index updates without full reconstruction.

To evaluate performance, a comparative experiment with the DGST method was conducted on subsets of a DNS name corpus ranging from 100 to 600 million characters under identical computational resources. RD-GST achieves a speedup of 3.2–5.1× and a throughput of 5.1–5.5 million characters/s compared to 0.9–1.7 million characters/s for DGST. The performance advantage is accompanied by higher memory consumption, requiring proportional resource scaling as corpus size increases.

The practical efficiency of the online approach for real-time streaming text corpus processing has been demonstrated. All results are reproducible using open-source implementations under controlled experimental conditions.

Keywords: generalized suffix tree; distributed computing; streaming data; data structures; text search.

Стаття надійшла до редакції 15.01.2026.