УДК 004.075

Є. Я. Ваврук, З. Г. Мозіль
Національний університет “Львівська політехніка”
кафедра електронних обчислювальних машин

ВИБІР АЛГОРИТМУ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ ПЕРЕДАВАННЯ ДАНИХ У РОЗПОДІЛЕНІЙ СИСТЕМІ

Розглянуто типову структуру багаторівневої розподіленої системи, проаналізовано проблеми передавання даних та можливість використання графів для їх вирішення. Обрано критерії вибору алгоритму пошуку оптимального шляху. На основі запропонованих критеріїв обрано алгоритм Беллмана - Форда для пошуку оптимального шляху в графі. Використано принципи Dirty Flag, CSR (розріджений ряд) та визначено параметри співвідношення «час-пам'ять» для збільшення швидкодії алгоритму.

Ключові слова: пошук оптимального шляху, розподілена система, алгоритм Беллмана-Форда, Dirty Flag, CSR.

E. Vavruk, Z. Mozil
Lviv Polytechnic National University
Computer Engineering Department

SELECTION OF OPTIMAL PATH FINDING ALGORITHM FOR DATA TRANSMITION IN DISTRIBUTED SYSTEMS

Considered typical structure of the multilevel distributed system, the data transmission problems are analyzed and the graphs can be used for their solution. The criteria for choosing the optimal path search algorithm are chosen. Based on the proposed criteria, the Bellman-Ford algorithm is chosen to find the optimal path in the graph. Used principles of Dirty Flag, CSR (Compressed Sparse Row) and defined time-memory relationship parameters to increase the speed of the algorithm.

Keywords: search of the optimal path, distributed system, Bellman-Ford algorithm, Dirty Flag, CSR.

Вступ

Однією з основних задач, які розв’язуються при проектуванні розподілених систем опрацювання даних, є забезпечення швидкості обміну між вузлами системи. Швидкість залежить від структури самої системи, протоколів обміну та вибраних алгоритмів передавання інформації. Причому в багатьох випадках саме алгоритми є ключовими при визначенні швидкодії системи.

Тому актуальною є проблема вибору алгоритму, який забезпечує оптимальний шлях передавання даних. Для цього розглянуто базові алгоритми пошуку оптимального шляху в графах, визначено основні критерії пошуку та обрано алгоритм.

Огляд літературних джерел

Розподілена система – це набір незалежних пристроїв, з'єднаних між собою в єдину систему. Розподілені системи структурно поділяються на однорівневі та багаторівневі. Однорівневі – системи, в яких усі вузли з'єднано. Багаторівневі – системи, в яких між окремими вузлами відсутній прямий зв'язок.

Недолік багаторівневих систем у тому, що передавання даних між вузлами системи є значно повільнішим ніж в однорівневій системі.

Оскільки кожен вузол багаторівневої розподіленої системи має прямий зв'язок принаймні з одним вузлом системи, відповідно, є можливість передавати дані між будь-якою парою вузлів через проміжні вузли.

Для передавання даних між двома вузлами можна використати цілу множину комбінацій проміжних вузлів. Для визначення оптимальної комбінації вузлів використано графи та алгоритми пошуку оптимального шляху в графах.

Багаторівневу систему зображено на рис. 1.

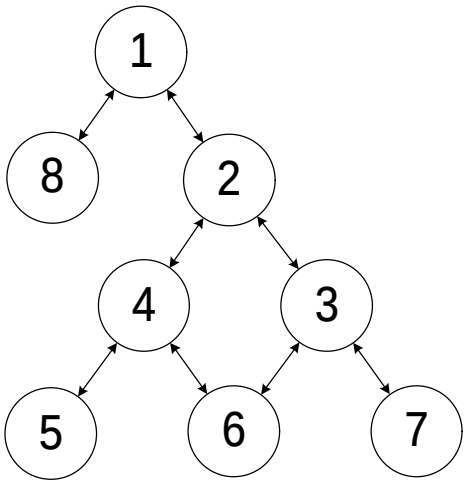

Рис. 1. Багаторівнева розподілена система

Ефективність роботи системи залежить від таких параметрів [5]:

- розмір – кількість вершин у графі;
- кількість зв'язків – сумарна кількість зв'язків між усіма вузлами графу;
- діаметр – довжина найкоротшого шляху між двома найвіддаленішими вузлами;
- порядок вузла – кількість зв'язків для конкретного вузла;
- зв'язаність – можливість знайти шлях між двома вершинами.

Вирішення проблеми передавання даних у багаторівневій розподіленій системі зводиться до розв'язання задачі пошуку найкоротшого (оптимального) шляху.

Ця задача є однією з основних задач, які розв'язуються за допомогою графів.

Існує узагальнений алгоритм пошуку оптимального шляху в графі, суть якого полягає в наступному: у зваженому графі вибирають усі можливі комбінації проходу з початкового до кінцевого вузла, після чого для кожної комбінації визначають сумарну вагу усіх ребер та серед усіх комбінацій обирають ту, яка має найменшу вагу.

Використання узагальненого алгоритму не є оптимальним через складність обчислень. Для пошуку оптимального шляху в графі існують алгоритми, перелік яких наведено в табл. 1.

Дані алгоритми пошуку оптимального шляху поділяють на три категорії:

- визначення оптимального шляху між парою вузлів;
- визначення оптимального шляху між всіма парами вузлів;
- визначення оптимального шляху між заданим вузлом та усіма іншими вузлами.

Таблиця 1

Перелік алгоритмів пошуку оптимального шляху в графі

Назва алгоритму	Задачі алгоритму
Дейкстри	Один вхід
Беллмана–Форда	Один вхід. При тому ваги між вузлами можуть бути від'ємними
A*	Одна пара вузлів
Флойда–Воршелла	Усі пари вузлів
Джонсона	Усі пари вузлів. При тому ваги між вузлами можуть бути від'ємними
Террі	Одна пара вузлів
Хвильовий	Одна пара вузлів
Данцига	Усі пари вузлів

У [1] описано використання алгоритму Дейкстри в локаційних службах. У [2] описано модифікований алгоритм Дейкстри для обчислення шляху за допомогою паралельних обчислень. У [3] описано алгоритм пошуку шляху в протоколі RIP (Routing Information Protocol). Проте в цих роботах не достатньо проаналізовано шляхи збільшення швидкодії роботи алгоритмів та принципи ефективного використання пам'яті.

Постановка завдання

Для вирішення проблеми вибору алгоритму оптимального шляху передавання даних у розподіленій системі потрібно:

- обрати структуру системи, для якої буде проводитися дослідження;
- визначити критерії відбору алгоритмів пошуку оптимального шляху;
- обрати алгоритм за заданими критеріями;
- дослідити роботу алгоритму.

Вибір алгоритму пошуку оптимального шляху передавання даних у розподіленій системі

Побудова структури системи

Приймемо, що вагою ребра графу є затримка передавання даних між вузлами (час передавання одного байта даних, мс), що дасть змогу визначити оптимальний шлях навіть у разі перевантаження деяких вузлів.

Запропоновано трирівневу структуру системи, яка містить 8 вузлів. Між будь-якими вузлами є шлях, який проходить не більше ніж через 2 проміжні вузли. Граф-схему системи наведено на рис. 2.

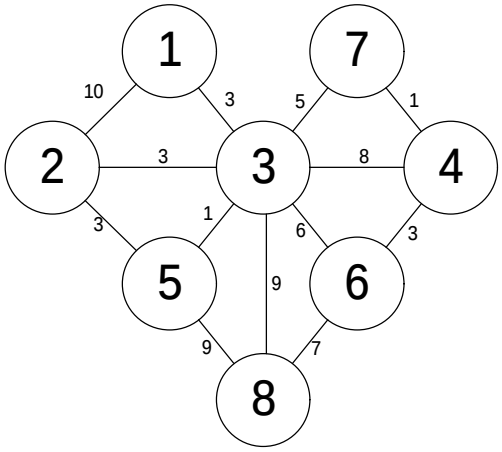

Рис. 2. Граф-схема системи

Визначення критеріїв відбору алгоритму оптимального шляху

Часова складність алгоритмів Флойда–Воршелла, Джонсона та Данциги є $O(N^3)$. Порівняно з іншими алгоритмами пошуку оптимального шляху, а саме алгоритмами пошуку шляху між парою вузлів та між заданим вузлом та усіма іними вузлами, часова складність є значно більшою. Тому використовувати такі алгоритми не ефективно.

Алгоритм A* використовує евристичну функцію для зменшення часової складності; цей алгоритм не гарантує визначення оптимального шляху між вузлами. Тому використання такого алгоритму не ефективне.

Як раніше було згадано: вага вузла – це затримка передавання даних між сусідніми вузлами, тому алгоритм повинен працювати з графами, в яких ваги між сусідніми ребрами можуть відрізнятися суттєво.

Окрім вищенаведених критеріїв, також потрібно, щоб була можливість обчислювати відстань між вузлами незалежно один від одного, для пришвидшення швидкодії пошуку оптимального шляху. Тобто має бути можливість виконувати алгоритм для різних пар вузлів паралельно.

Отже, алгоритм пошуку оптимального шляху повинен відповідати таким критеріям:

1. Розв'язувати задачу пошуку оптимального шляху з однією парою вузлів або між заданим вузлом і усіма іншими вузлами.
2. Не використовувати евристичних функцій.
3. Мати можливість працювати з графами, в яких ваги сусідніх ребер можуть суттєво відрізнятися.
4. Відстані між різними парами вузлів можуть обчислюватися паралельно.

Вибір алгоритму пошуку оптимального шляху

Враховуючи наведені критерії перелік алгоритмів (див. табл. 1) можна зменшити, оскільки:

– алгоритми Флойда–Воршелла, Джонсона та Данцига не відповідають першому критерію;

– алгоритм A* використовує евристичну функцію, тому не відповідає другому критерію:

– хвильовий алгоритм може працювати лише з графами, які мають одиничну довжину, відповідно, алгоритм не відповідає третьому критерію;

– обчислювати відстань до наступного вузла в алгоритмі Террі можна лише тоді, коли обчислено відстань до попереднього вузла. Тому алгоритм не відповідає четвертому критерію.

Отже, вибраним критеріям задовольняють алгоритми Дейкстри та Беллмана–Форда.

На відміну від алгоритму Дейкстри, алгоритм Беллмана–Форда допускає від'ємні ваги ребер в графі. Окрім цього, складність алгоритму Беллмана–Форда є $O(VE)$ (V – кількість вершин, E – кількість ребер), коли складність алгоритму Дейкстри в класичній реалізації є $O(V^2E)$. Отже, алгоритм Беллмана–Форда обрано для реалізації передавання даних у розподіленій системі.

Опис алгоритму Беллмана–Форда

В алгоритмі використано принцип релаксації: при кожному відвідуванні вершини до неї записується відстань до найближчого вузла. Відповідно, в кінці роботи алгоритму кожен пройдений вузол міститиме відстань до найближчого сусіднього вузла.

Нехай потрібно знайти шлях із точки S до точки F. Вузол, для якого обчислюється відстань – V. Обираємо вузол, який є сусіднім до поточного вузла, позначимо його U. Після цього відбувається процес релаксації: якщо distance(S, V) + distance(V, U) < distance(S, U), тоді оновлюється відстань від S до U (distance(S, U) = distance(S, V) + distance(V, U)). Алгоритм гарантує визначення оптимального шляху до кожного вузла графу за (V – 1) ітерацій, де V – кількість вершин у графі.

Псевдокод алгоритму наведено нижче.

```
function BellmanFord(list vertices, list edges, vertex source)::distance[],
predecessor[]
   // ініціалізація графу
   for each vertex v in vertices:
```

```
    distance[v] := inf
    predecessor[v] := null
distance[source] := 0

// релаксація на кожній вершині
for i from 1 to size(vertices)-1:
    for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
            distance[v] := distance[u] + w
            predecessor[v] := u

return distance[], predecessor[]
```

Використання принципу Dirty Flag в алгоритмі Беллмана–Форда.

Принцип Dirty Flag полягає у тому, що до алгоритму додається ще одна змінна, яка містить дані про факт зміни масиву відстаней. За кожної зміни відстані ця змінна встановлюватиметься в «1». Якщо перед обчисленням відстані до вузла змінна залишилася в «1», то робота алгоритму продовжується, якщо змінну встановилася в «0», то роботу алгоритму завершено.

Як видно з псевдокоду алгоритму, масив відстаней змінюється кожної ітерації. Тому можна зробити висновок: алгоритм завершив роботу, якщо не відбулося зміни масиву відстаней. Причому, робота алгоритму може бути завершена навіть коли алгоритм не пройшов усіх вузлів, наприклад, за наявності «висячих» вершин.

На рис. 3 наведено граф-схему для виконання алгоритму Беллмана–Форда. Початковий вузол – вершина A. Якщо передавати дані з правого у лівий бік, то алгоритм виконається за 4 ітерації. Якщо ж з лівого боку у правий, то алгоритм виконається за одну ітерацію, і далі масив відстаней не буде оновлюватися.

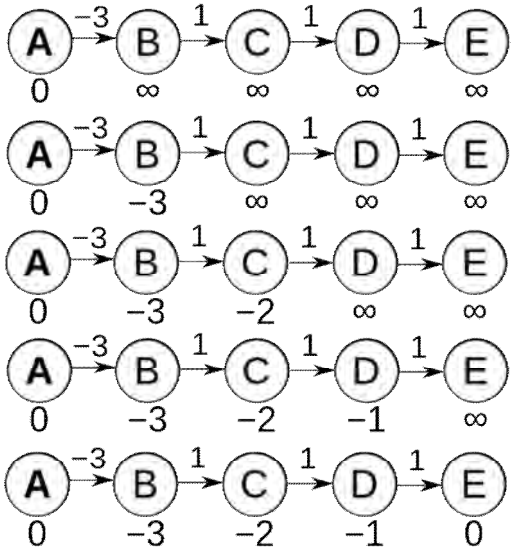

Рис. 3. Граф-схема для виконання алгоритму Беллмана–Форда

Використання CSR (Comressed sparse row) замість матриці суміжності

Для збереження графів зазвичай використовують матриці суміжності. Це матриці N*N, де N – кількість вершин у графі. На перетині вершин, які з'єднано ребром, встановлюється «1». Якщо вершини не з'єднано, тоді встановлюється «0».

У табл. 2 наведено матрицю суміжності графу системи (див. рис. 2)

Таблиця 2

Дані графу, записані матрицею суміжності

N	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	0	1	0	0	0
3	1	1	0	1	1	1	1	1
4	0	0	1	0	0	1	1	0
5	0	1	1	0	0	0	0	1
6	0	0	1	1	0	0	0	1
7	0	0	1	1	0	0	0	0
8	0	0	1	0	1	1	0	0

Аналіз таблиці показує, що для обчислення обсягу пам'яті, яка потрібна для зберігання даних графу за допомогою матриці суміжності, використовують формулу (1).

$$K = V^2 M, \tag{1}$$

де V – кількість вершин у графі; M – обсяг пам'яті для збереження інформації про одне ребро (біт).

Обсяг пам'яті залежить від кількості вершин у графі і збільшується в квадратичній залежності.

Для зменшення обсягів пам'яті із збереженням графів використаємо CSR (розріджений ряд), який потребує зберігання трьох масивів даних. Перший масив – індекси вершин (indeces), другий масив – сусідні вершини (ptr), третій масив – ваги ребер.

Перший масив містить індекси для масиву сусідніх вершин. Для того, щоб дізнатися, які вершини є сусідніми до вершини «i», потрібно дістати усі значення з другого масиву (ptr) з індексами від indeces[i] до indeces[i + 1] не включно. Значення індексів ваг відповідають індексам сусідніх вершин.

У табл. 3 наведено дані графу системи, записаного способом CSR (рис. 2)

Таблиця 3

Дані графу, записаного способом CSR

Indeces	1	2	3	4	5	6	7	8
	1	3	6	12	15	18	21	23

N	1	2	3	4	5	6	7	8	9	10	11	12	13
Ptr	2	3	1	3	5	1	2	4	5	6	7	3	7
N	14	15	16	17	18	19	20	21	22	23	24	25	
Ptr	6	2	3	8	3	4	8	3	4	3	5	6	

Аналіз таблиці показує, що для обчислення обсягу пам'яті, який потрібен для збереження даних графу способом CSR, використовують формулу (2).

$$K = M(V + E), \tag{2}$$

де M – обсяг пам'яті для збереження інформації про одне ребро (біт); V – кількість вершин графу; E – кількість ребер графу.

Графи, записані способом CSR, займають менше пам'яті, ніж графи, записані матрицею суміжності. Наприклад, якщо для збереження інформації про одну вершину потрібно $M = 8$ бітів, то із використанням матриці суміжності для зберігання графу (див. рис. 2) потрібно $K = V^2 M = 8^2 * 8 = 512$ біти, а за використання способу CSR потрібно $K = M(V \mid E) = 8 * (8 \mid 25) = 264$ біти.

Співвідношення час-пам'ять

Часова складність алгоритму Беллмана–Форда залежить від кількості вершин у графі. Для зменшення кількості обчислень можна використати співвідношення «час–пам'ять», за більшого використання пам'яті час виконання зменшується.

Частину даних, які обчислили алгоритмом про відстані до вершин графу можна зберігати у пам'яті. Тоді за необхідності знайти шлях з того самого вузла шлях може бути завантажений з пам'яті, а не обчислений заново.

Дані, обчислені за алгоритмом Беллмана–Форда, можна зберігати частково, що дасть змогу підібрати оптимальне співвідношення час–пам'ять для конкретної системи.

Висновки

У результаті проведеного аналізу розроблено граф розподіленої системи, обрано критерії відбору алгоритму пошуку оптимального шляху в графі, алгоритм Беллмана–Форда.

Визначено вирази для обчислення обсягів пам'яті, які потрібні для зберігання даних графів за допомогою методу матриці суміжності та методу CSR. З використанням матриці суміжності обсяг пам'яті квадратично залежить від кількості вершин у графі.

За принципом Dirty Flag можна припинити роботу алгоритму, коли опитмальний шлях вже знайдено, але відстані записано ще не для усіх вершин. Також проаналізовано співвідношення «час-пам'ять», при використанні якого обчислені шляхи зберігаються в пам'яті, та за потреби вони заново не обчислюються.

Надалі заплановано розробити генератор графів за заданими критеріями, граф-схему алгоритму Беллмана–Форда за технологією CUDA. Побудовані графи використати для пошуку оптимального шляху передавання даних та дослідити швидкодію та обсяги пам'яті для різних типів графів.

1. Dolinskaya I. Optimal Path Finding in Direction, Location and Time Dependent Environments / Irina Dolinskaya. – Evanston, 2012. – 33 с. – (Northwestern University). 2. Pradesh M. Modified Dijkstra's Algorithm for Dense Graphs / Madhua Pradesh. – Bhopal, India, 2016. – 9 с. – (Maulana Ajad National Institute of Technology). 3. Krianto S. Bellman Ford algorithm in Routing Information Protocol / Sulaiman Krianto. – Indonesia, 2018. – 10 с. – (Universitas Prima Indonesia). 4. Aksak N. Vykorystannia alhorytmiv poshuku naikorotshoho shliakhu na hrafakh (Using algorithms to find the shortest path on the graphs) / Nikolay Aksak. – Kharkiv, 2004. – 10 с. 5. Dunets R. Topolohiia kompiuternykh system (Topology of computer systems) / Roman Dunets. – Lviv, 2007. – 48 с. – (Lviv Polytechnic National University).