**Національний технічний університет України**

**«Київський політехнічний інститут»**

факультет інформатики та обчислювальної техніки

кафедра обчислювальної техніки

Курсовий проект

*з дисципліни:* **«Проектування комп’ютерних систем»**

|  |  |
| --- | --- |
| **Виконав**: | Мазур Роман Федорович |
| **Група:** | ІО-52 |
| **Перевірили**: | Луцький Г. М. \_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис керівника)  Кулаков Ю. А. \_\_\_\_\_\_\_\_\_\_\_\_\_  (підпис керівника) |

2009 р.

ТЕХНІЧНЕ ЗАВДАННЯ

1. Синтезувати оригінальну масштабовану топологічну організацію. Представити рисунок організації.
2. Визначити топологічні характеристики синтезованої організації. Проаналізувати такі параметри, як: ступінь, діаметр, середній діаметр, вартість, трафік, зв’язність.
3. Порівняти синтезовану топологію з гіперкубом.
4. Відобразити топологію в гіперкуб.
5. Розробити відмовостійкі алгоритми для чотирьох видів маршрутизації для синтезованої топології:
   1. один до одного,
   2. усі до всіх з персональним призначенням,
   3. один до всіх,
   4. multicast.
6. Розглянути розв’язання прикладної задачі на синтезованій топологічній організації.

ПОЯСНЮВАЛЬНА ЗАПИСКА

**Зміст**

[1 Синтез топології 6](#_Toc250947996)

[1.1 Принципи побудови 6](#_Toc250947997)

[1.2 Розмір та нумерація топології 8](#_Toc250947998)

[1.3 Закони формування зв’язків у топології 9](#_Toc250947999)

[1.4 Базові функції для роботи з топологією 9](#_Toc250948000)

[2 Аналіз топології 11](#_Toc250948001)

[2.1 Визначення топологічних характеристик 11](#_Toc250948002)

[2.2 Порівняння топології з гіперкубом 12](#_Toc250948003)

[3 Відображення топології у гіперкуб 16](#_Toc250948004)

[4 Маршрутизація 17](#_Toc250948005)

[4.1 Один до одного 18](#_Toc250948006)

[4.2 Усі до всіх з персональним призначенням 23](#_Toc250948007)

[4.3 Один до всіх 28](#_Toc250948008)

[4.4 Multicast 30](#_Toc250948009)

[5 Завантаження прикладної задачі 33](#_Toc250948010)

[5.1 Генетичні алгоритми в еволюції 33](#_Toc250948011)

[5.2 Завантаження у топологію 35](#_Toc250948012)

[Висновки 37](#_Toc250948013)

[Додаток 38](#_Toc250948014)

[Лістинг програми 38](#_Toc250948015)

1 Синтез топології

1.1 Принципи побудови

В основу структурної організації топології було вирішено покласти гіперкуб у зв’язку з тим, що його побудова дає можливість завантажувати дуже великий спектр прикладних задач. Для того, щоб збільшити відмовостійкість топології основою його системи числення була вибрана тріскова замість двійкової, класичної. Цікавою особливістю такої топології є те, що трійковий гіперкуб 2-го порядку вироджується у меш-топологію 3х3 (рис. 1.1).



Рис. 1.1 Трійковий гіперкуб 2-го порядку

Основною вадою гіперкубічних топологій є їхній високий степінь, який визначається алгебраїчно степеневою функцією (*аN*). У нашому випадку ріст значення степеня буде стрімкішим за класичний через більше значення основи системи числення. Це, у свою чергу, спричиняє швидке зростання вартості системи. Таким чином, необхідно зупинити зростання степеня топології.

Для мінімізації вартості системи вершиною гіперкуба виступає не окремий вузол, а кластер з кількістю вузлів рівною подвоєному значенню порядку системи. Оскільки, в такому випадку кількість вузлів у кластері буде рівною степеню вершини 3-вого гіперкуба, то кожен з вузлів матиме один зв’язок з іншим кластером. Найпростіший варіант зв’язків у кластері – кільце (рис. 1.2).

|  |  |
| --- | --- |
| а) | б) |
| Рисунок 1.2 а) вершина трійкового гіперкуба 3-го порядку,  б) зменшення степеня вершини | |

Зі збільшенням порядку топології все більше значення матимуть параметри зв’язності внутрішнього кластера, а не зовнішнього гіперкуба. Тому, щоб збільшити зв’язність у кластері, введено додаткове ребро з «протилежною» вершиною. У результаті отримано топологічну організацію, показану на рисунках 1.3 та 1.4.

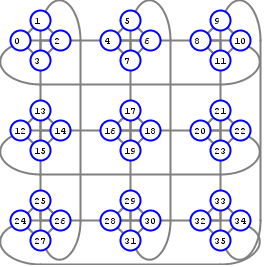


Рисунок 1.3 Топологічна організація 2-го порядку

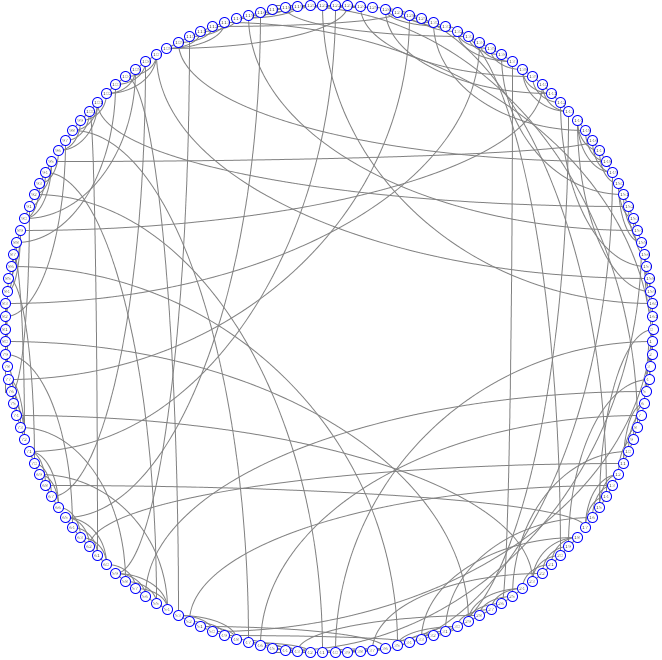


Рисунок 1.4 Топологічна організація 3-го порядку

1.2 Розмір та нумерація топології

Топологічна організація формується з кластерів. Їхня кількість визначається кількістю вершин трійкового гіперкуба відповідного порядку. Тобто, для організації з порядком *d* матимемо 3*d* кластерів. Кожен кластер має розмір (кількість вузлів, що його формують) рівний степеню вершини трійкового гіперкуба, тобто .

Кожен вузол у топології має наскрізний номер, однак в алгоритмах, звісно, часто використовуються поняття номера кластера та номера вузла у кластері. Перший може бути визначений, як частка від ділення наскрізного номера на розмір кластера, а другий, як остача цього ділення.

1.3 Закони формування зв’язків у топології

Як уже було сказано, кожен вузол у синтезованій топології має 4 зв’язки: три з них – зв’язки всередині кластера, і ще один – ребро зовнішнього трійкового гіперкуба. Нехай, *d* – порядок топології, *cs* – розмір кластера, *і* – номер вузла (наскрізний), *N* – кількість вершин у топології. Тоді чотири зв’язки *і*-го вузла можуть бути описані наступним чином ().

1. – зв’язок усередині кластера, за годинниковою стрілкою, рисунок 1.5 а).
2. – зв’язок усередині кластера, проти годинникової стрілки, рисунок 1.5 б).
3. – зв’язок усередині кластера з «протилежною» вершиною, рисунок 1.5 в).
4. – зв’язок з відповідною вершиною сусіднього кластера, рисунок 1.5 г).



Рисунок 1.5 Чотири зв’язки між вузлами

1.4 Базові функції для роботи з топологією

Далі наведемо основні функції, що можуть бути корисними при роботі з синтезованою топологічною організацією. В усіх формулах *d* – порядок топології, *і* та *j* – номера вузлів.

|  |  |
| --- | --- |
| Таблиця 1.1 Базові функції | |
| Кількість вузлів |  |
| Розмір кластера |  |
| Номер кластера |  |
| Номер вузла у кластері |  |
| Номер відповідної вершини сусіднього кластера | **int** axis = ni % d;  **int** sourceConnector = ni / d;  **int** nearDigit = *getDigit*(ci, axis);  nearDigit += sourceConnector == 0 ? -1 : 1;  nearDigit %= 3;  **if** (nearDigit < 0) { nearDigit=3+nearDigit; }  **int** nearCluster = *setDigit*(ci,axis,nearDigit);  **return** nearCluster \* cs + (ni + d) % cs; |
| Отримання масиву сусідніх вершин | **int** startNode = i / cs \* cs;  **return** **new** Integer[] {  startNode + (i + 1) % cs,  startNode + (i > 0 ? (i - 1) % cs : cs - 1),  startNode + (i + d) % cs,  *getNearClusterConnection*(i, d)  }; |

2 Аналіз топології

2.1 Визначення топологічних характеристик

Розглянемо наступні топологічні характеристики:

* N – кількість вузлів;
* S – степінь;
* D – діаметр;
* Dсер – середній діаметр;
* C – вартість;
* T – трафік;
* глобальна зв’язність.

Дані параметри залежатимуть від порядку топологічної організації, тому розглянемо закони, за якими вони можуть бути визначені.

У таблиці 2.1 наведено дані показники для різного порядку топології.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблиця 2.1 Топологічні характеристики | | | | | | |
| d | N | S | D | Dсер | T | C |
| 2 | 36 | 4 | 5 | 3,11 | 2,08 | 540 |
| 3 | 162 | 4 | 9 | 5,27 | 3,52 | 4374 |
| 4 | 648 | 4 | 11 | 6,99 | 4,66 | 21384 |
| 5 | 2430 | 4 | 15 | 9,17 | 6,11 | 109350 |

Кількість вузлів має дуже стрімке зростання при масштабуванні та визначається за формулою

Степінь є геометричною характеристикою, що характеризує локальну зв’язність організації та визначається як максимальне число ребер, інцидентних одній вершині. Для синтезованої топологічної організації степінь є константою, що було однією із задач синтезу.

Діаметр топології є мінімальною відстанню між максимально віддаленими вершинами в топології. Його можна визначити через матрицю зв’язності. Для синтезованої організації діаметр визначається формулою

Середній діаметр показує усереднену відстань між вузлами у топології та обчислюється за допомогою матриці відстаней ().



Вартість визначається як мультиплікативний критерій діаметра, кількості вузлів та степеню.

Вартість показує, на скільки раціонально використовувати топологію при заданих параметрах, тобто фактично визначає співвідношення «ціна-якість».

Трафік (або густина трафіку) визначається за формулою

Дана характеристика показує ефективність використання ребер у топології. Чим ближче її значення до 1, тим ефективнішим є потенційне використання.

Під глобальною зв’язністю будемо розуміти число ребер, які необхідно розрізати для того, щоб отримати два незв’язаних між собою підграфа. Для того, щоб виконати таку операцію, необхідно розрізати ті ребра, які зв’язують між собою кластери з різними координатами на одній осі. Таким чином, дана величина буде рівною .

2.2 Порівняння топології з гіперкубом

Наведемо таблицю з топологічними характеристиками для гіперкуба.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблиця 2.2 Топологічні характеристики гіперкуба | | | | | |
| N | S | D | Dсер | T | C |
| 32 | 4 | 5 | 2,58 | 1,29 | 640 |
| 64 | 5 | 6 | 3,05 | 1,22 | 1920 |
| 128 | 6 | 7 | 3,53 | 1,18 | 5376 |
| 256 | 7 | 8 | 4,02 | 1,15 | 14336 |
| 512 | 8 | 9 | 4,51 | 1,13 | 36864 |
| 1024 | 9 | 10 | 5 | 1,11 | 92160 |
| 2048 | 11 | 11 | 5,7 | 1,11 | 128410 |

Для зручності аналізу побудуємо порівняльні графіки для характеристик гіперкуба та нашої топології.

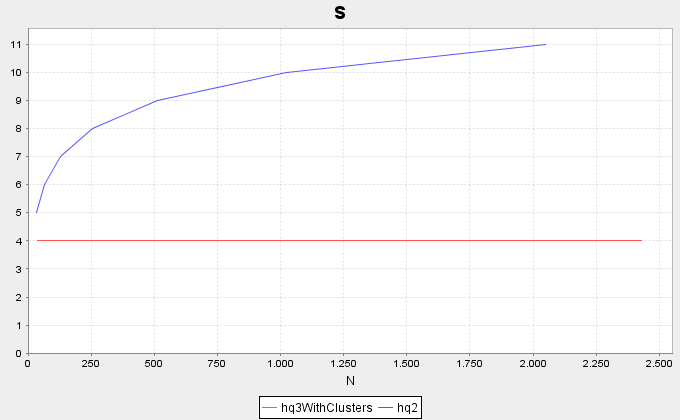


Рисунок 2.1 Степінь

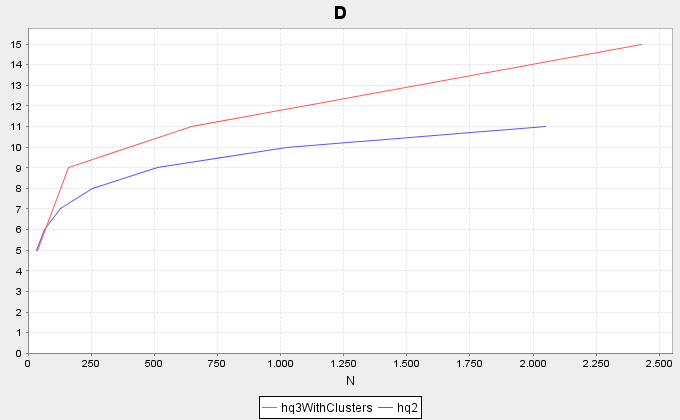


Рисунок 2.2 Діаметр

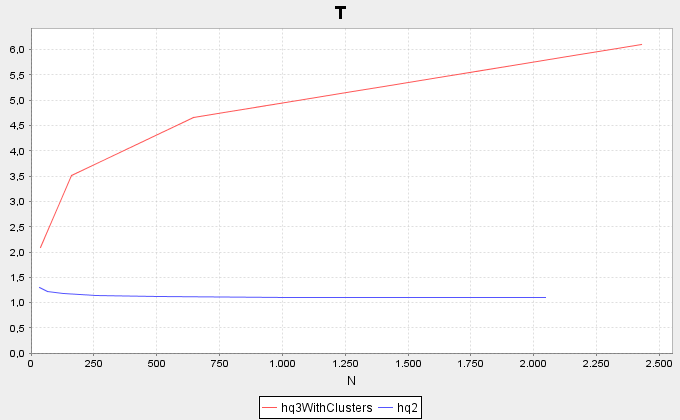


Рисунок 2.3 Трафік

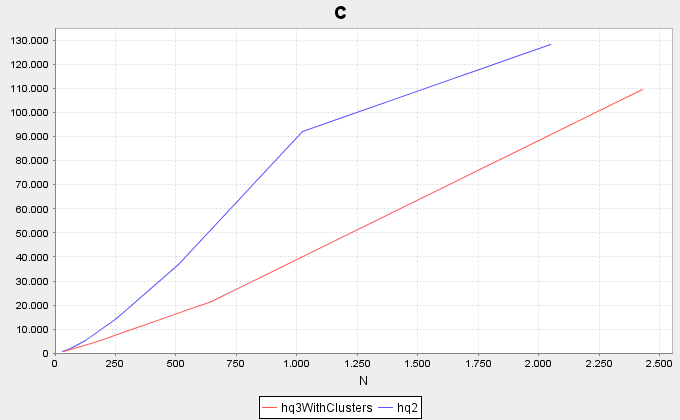


Рисунок 2.4 Вартість

Таким чином, наша топологія виконує свою задачу, роблячи степінь у гіперкубічної організації константою, що зумовлює повільніше зростання вартості. При цьому зберігається логарифмічний характер росту діаметра. Ціною, яку необхідно платити за таке зниження вартості системи, є зменшення ефективності використання ребер топології – вони отримують значно більше завантаження, що впливає не роботу алгоритмів маршрутизації, особливо на варіант «усі до всіх».

3 Відображення топології у гіперкуб

Для відображення синтезованої топологічної організації у гіперкуб природно представити кожен з кластерів як одну вершину гіперкуба. У результаті отримаємо трійковий гіперкуб (рисунок 3.1). Таким чином задача відображення зводиться до відображення гіперкуба з системою числення 3 до двійкового.

|  |  |
| --- | --- |
|  |  |

Рисунок 3.1 Перехід до трійкового гіперкуба

У трійковому гіперкубі на кожній з осей маємо по три вершини. Це створює великі незручності при відображенні у двійкову систему. Для розв’язання даної проблеми можна збільшити кількість вершин на кожній з осей до чотирьох (рисунок 3.2). У такому випадку ми отримаємо двійкову гіперкубічну організацію.

|  |  |
| --- | --- |
|  |  |

Рисунок 3.2 Перехід до n-вимірної меш-топології

4 Маршрутизація

Маршрутизація – це задача або процес пошуку шляху передачі даних від відправника до приймача. Це дуже важливий елемент в роботі усієї системи, оскільки розв’язанні задач на багатокомп’ютерних системах важливе місце посідає обмін інформацією між вузлами. Необхідно забезпечити задовільні час та надійність передачі даних для того класу задач, який має розв’язуватися на синтезованій топологічній організації. Час, який витрачається на передачу інформації з одного вузла в інший відіграє дуже важливу роль, оскільки чим швидше певний вузол отримає необхідні дані, тим швидше він почне їхню обробку. Також при розробці алгоритмів маршрутизації необхідно отримати в результаті такий принцип взаємодії вузлів, який забезпечував би найбільш ефективне використання потенційних можливостей системи. Тому алгоритми мають бути універсальними та застосовуватися для будь-якого масштабу топологічної організації.

При розробці алгоритмів враховувався фактор динамічної відмови вузлів (тобто в ході передачі даних між відправником та приймачем). Вважається, що кожен з вузлів має можливість перевірки стану кожного зі своїх лінків.

Варто відзначити, що всі алгоритми передбачають, що кожен з каналів, якими з’єднані вузли системи працюють у дуплексному режимі, тобто дозволяють одночасну передачу в обох напрямках, а вузол може одночасно приймати дані з одного лінка та передавати на інший, або той самий.

Вважатимемо, що маршрутизація заснована на пакетах. Тобто пакет є атомарною (неподільною) одиницею передачі даних. Узагальнюючи поняття пакета, наведемо його структуру (рисунок 4.1).



Рисунок 4.1 Загальна структура пакета

Пакет включає в себе наступні поля:

* id – унікальний ідентифікатор пакета у системі;
* type – тип передачі (один до одного, усі до всіх і т.д.);
* src – адреса відправника;
* dst – адреса приймача;
* path – поле, у якому фіксується пройдений шлях (має змінну величину та змінний формат для різних типів передачі);
* data – користувацькі дані.

При розгляді ситуацій з відмовами вважається, що відмовити може лінк на вузлі. Тоді вузол, що повністю відмовив, розглядається, як множина відмовивши лінків, що ведуть до даного вузла. Це забезпечує більш ефективну роботу алгоритмів.

Час, який наводиться для маршрутизацій, не враховує наявність вузлів, зо відмовили (у такому випадку він, як правило, зростає).

4.1 Один до одного

Особливості топологічної організації породжують два підтипи маршрутизації: маршрутизація в кластері та маршрутизація в зовнішньому гіперкубі.

Варто відзначити, що інформацію про відмови вузлів містять лише їхні сусіди, і рішення про обхід недосяжного вузла відбувається в динаміці. Такий підхід вимагає механізму уникнення циклів.

Для цього зберігається інформація про вже відвідані вузли. Вона може передаватися в заголовку повідомлення, що звісно, збільшує об’єм службової інформації. Однак великий об’єм буде передаватися лише у випадку великої кількості відмов (при збільшення довжини маршруту). Тому у випадках, коли даний момент стає критичним, варто скористатися технікою віртуальної комутації каналів, при якій встановити з’єднання, за допомогою маршрутизації спеціального пакета описаними способами, а потім передавати користувацькі дані за вже встановленим шляхом.

*Маршрутизація у кластері*

Нехай cnIndex – номер поточного вузла в кластері (cnIndex , тут і далі d – порядок гіперкуба).

Тоді зв’язані вузли можуть бути записані масивом

[(cnIndex + 1) % cs, (cnIndex + d) % cs, (cnIndex – 1) % cs]

З цих трьох вузлів вибирається той, який найменш віддалений від вузла призначення, доступний та ще не був відвіданий.

Якщо такого сусіда не знайдено, вузол призначення вважається недосяжним в рамках поточного кластера. У сусідній кластер, при цьому, пакет не передається, оскільки це може призвести до довгого подальшого маршруту, який, ти не менше, може й не досягнути вузла призначення.

Для обчислення відстані між вузлами в кластері використовується наступна функція:

**int** getInClusterDistance(**final** **int** i, **final** **int** j, **final** **int** d) {

**int** dif = j - i;

**if** (dif == 0) { **return** 0; }

**int** sign = dif > 0 ? 1 : -1;

dif = Math.*abs*(dif);

**int** cs = d << 1;

**int** next = 0;

**if** (dif > d) {

next = (i + (dif < d + (d >> 1) ? d : -sign)) % cs;

} **else** {

next = (i + (dif > d >> 1 ? d : sign)) % cs;

}

**if** (next < 0) { next = cs + next; }

**return** *getInClusterDistance*(next, j, d) + 1;

}

*Міжкластерна маршрутизація*

Міжкластерна маршрутизація здійснюється в трійковому гіперкубі. При чому кожен вузол кластера має один лінк даного гіперкуба.

Нехай currentNode – номер поточного вузла, destNode – номер вузла призначення.

Тоді

currentCluster = currentNode / 2d – номер поточного кластера,

destCluster = destNode / 2d – номер кластера, якому належить вузол призначення.

Знаходяться вісі гіперкуба, в яких відрізняються координати у поточного та цільового кластера. Маючи дані вісі, легко знаходяться сусідні кластери, через які може відбуватися передача даних. Сусідніми кластерами є ті, що відрізняються в координатах лише в одній осі.

Далі знаходяться вузли, що належать поточному кластеру та мають лінки до сусідніх кластерів (їх назвемо місцями стрибка). Ці дані можуть бути обчислені при ініціалізації кожного з вузлів системи та записані в локальну пам’ять для уникнення зайвих обчислень.

Знайдений масив вузлів сортується за зростанням відстані до поточного вузла (за допомогою функції getInClusterDistance). До нього додаються вузли, протилежні до знайдених.

Далі отриманий список переглядається у встановленому порядку та вибирається перше місце стрибка, яке є досяжним, а цільова вершина після стрибка ще не була відвідана.

Якщо ми знаходимося у вибраному вузлі, то стрибок відбувається, коли цільова вершина є доступною. Інакше розглядається наступний варіант.

Якщо після розгляду усіх елементів попереднього масиву рішення про перехід не було прийнято, то це означає, що варто спробувати виконати передачу по осі, де немає відмінностей в координатах. Вибираються всі місця стрибків, що не розглядалися раніше, алгоритм повторюється для них.

Якщо рішення про перехід не було прийнято й зараз, то вузол призначення вважається недосяжним.

Час маршрутизації буде рівний:

Розглянемо приклад маршрутизації на топологічній організації третього порядку (рисунок 4.2). Повідомлення передається з 5-го в 100-ий вузол.

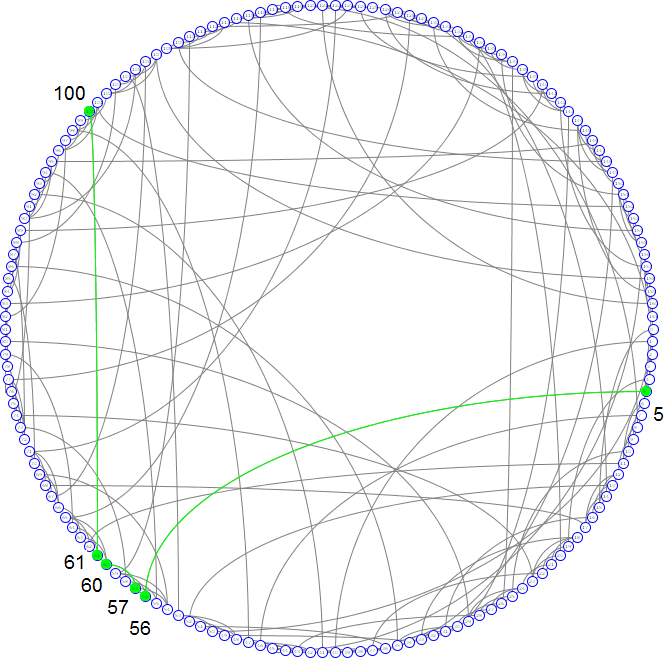


Рисунок 4.2 Приклад маршрутизації один до одного

Лог маршрутизації:

<Step1. [{5 -> 56: Jumping from 5 to 56.}]>

<Step2. [{56 -> 57: Look for 57 to jump.; go to 57}]>

<Step3. [{57 -> 60: Jumping from 57 to 60.}]>

<Step4. [{60 -> 61: Look for 61 to jump.; go to 61}]>

<Step5. [{61 -> 100: Jumping from 61 to 100.}]>

Розглянемо також приклад маршрутизації з вузлами, що відмовили (46 та 56).

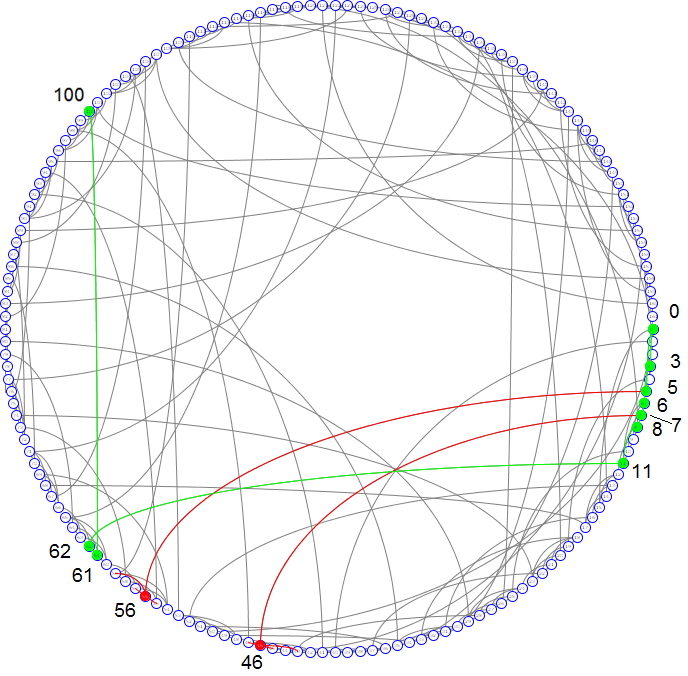


Рисунок 4.3 Приклад маршрутизації один до одного з вузлами, що відмовили

Лог маршрутизації:

<Step1. [{5 -> 0: Jump link to 56 is down.Look for 3 to jump.; go to 0}]>

<Step2. [{0 -> 3: Look for 3 to jump.; go to 3}]>

<Step3. [{3 -> 6: Jumping from 3 to 6.}]>

<Step4. [{6 -> 7: Look for 7 to jump.; go to 7}]>

<Step5. [{7 -> 8: Jump link to 46 is down.Look for 11 to jump.; go to 8}]>

<Step6. [{8 -> 11: Look for 11 to jump.; go to 11}]>

<Step7. [{11 -> 62: Jumping from 11 to 62.}]>

<Step8. [{62 -> 61: Look for 61 to jump.; go to 61}]>

<Step9. [{61 -> 100: Jumping from 61 to 100.}]>

*Приклад роботи програми*

Передача з 9-го у 26-ий вузол. Відмовили 35, 1 та 15.

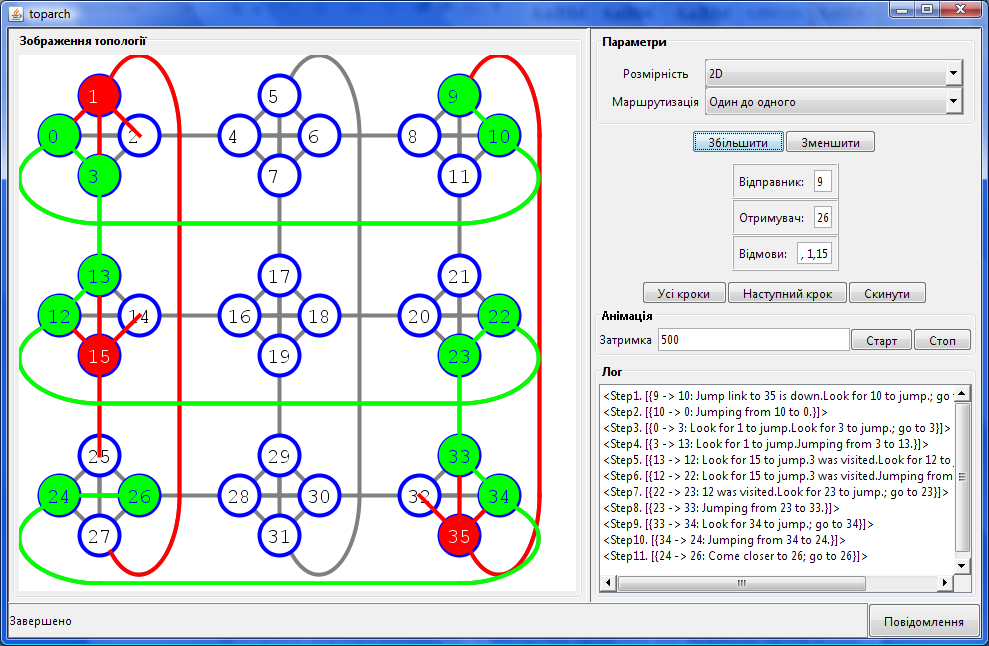


Рисунок 4.4 Приклад роботи програми при маршрутизації один до одного

4.2 Усі до всіх з персональним призначенням

Даний тип маршрутизації використовується для ефективного одночасного обміну між усіма вузлами ком’ютерної системи даними, необхідними для виконання обчислень.

Алгоритм маршрутизації має конвеєрний принцип. На кожному кроці у вузлі вибирається повідомлення, найбільш віддалене від свого вузла призначення. Для цього повідомлення, що мають бути відправлені тримаються в пріоритетній черзі, а пріоритет повідомлення визначається в залежності від віддалення до кінцевого вузла. Також при визначенні пріоритету перевага віддається транзитним пакетам, а не тим, відправником яких є поточний вузол. Така черга може бути реалізована за допомогою червоно-чорних дерев.

Кожен вузол у топології має чотири сусіди. На кожному кроці вибирається наступний сусід та повідомлення для відправки в цей вузол. У результаті на кожному кроці маємо пари вузлів, які обмінюються між собою повідомленнями. Існує 4 стратегії вибору партнера для обміну повідомленнями:

1. Передача в кластері за годинниковою стрілкою (partner = node + 1).
2. Передача в кластері протилежному вузлу (partner = node + d).
3. Передача в кластері проти годинникової стрілки (partner = node - 1).
4. Стрибок між кластерами (partner = near(node)).

Вибірка наступного вузла для відправки повідомлення відбувається аналогічно до роботи алгоритму один до одного. Це забезпечує обхід вузлів чи каналів, що відмовили. Функція,що визначає наступний вузол виглядає наступним чином.

**public** **static** **int** getNextNode(**final** **int** i, **final** **int** j, **final** **int** d, **final** RouteFilter filter) {

**if** (i == j) { **return** -1; }

**final** **int** cs = d << 1;

**final** **int** ci = i / cs, cj = j /cs;

**final** **int** clusterSource = i % cs;

RouteFilter filterWrapper = **new** RouteFilter() {

@Override

**public** **boolean** accept(**final** **int** clusterIndex) { **return** filter.accept(ci \* cs + clusterIndex); }

};

**if** (ci == cj) { // same cluster

**int** clusterDest = j % cs;

**int** res = *getNextRouteInCluster*(clusterSource, clusterDest, d, filterWrapper);

**if** (res < 0) { **return** -1; }

**return** ci \* cs + res;

} **else** { // between clusters

// find transition clusters

**int**[] axises = *compareClusters*(ci, cj, d);

**int**[] transitionClusters = **new** **int**[axises.length];

**for** (**int** k = 0; k < transitionClusters.length; k++) {

transitionClusters[k] = Utils.*setDigit*(ci, axises[k], Utils.*getDigit*(cj, axises[k]));

}

// define jump points to that clusters

List<Integer> jumpPoints = **new** ArrayList<Integer>(cs);

**for** (**int** cluster : transitionClusters) {

jumpPoints.add(Utils.*getNearInfo*(ci, cluster, d).getSource());

}

Collections.*sort*(jumpPoints, **new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer p1, Integer p2) {

**return** Utils.*getInClusterDistance*(p1, clusterSource, d) - Utils.*getInClusterDistance*(p2, clusterSource, d);

}

});

List<Integer> altJP = **new** LinkedList<Integer>();

**for** (Integer jp : jumpPoints) { altJP.add((jp + d) % cs); }

jumpPoints.addAll(altJP);

**for** (**int** k = 0; k < cs; k++) { **if** (!jumpPoints.contains(k)) { jumpPoints.add(k); } }

// try jump points

**for** (Integer njp : jumpPoints) {

**int** jpNode = cs \* ci + njp;

**int** nextNode = *getNearClusterConnection*(jpNode, d);

**if** (!filter.accept(nextNode)) { **continue**; } // we were there

**if** (i == jpNode) {

// jump

**return** nextNode;

} **else** {

// move in cluster[

**if** (!filter.accept(jpNode)) { **continue**; }

**int** res = *getNextRouteInCluster*(clusterSource, njp, d, filterWrapper);

**if** (res >= 0) { **return** ci \* cs + res; }

}

}

**return** -1;

}

}

Максимальний час маршрутизації рівний

Розглянемо приклад маршрутизації на топології 2-го порядку з відмовивши вузлом під номером 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Крок 1 – рисунок 4.5 а) | | Крок 2 – рисунок 4.5 б) | |
| {0 -> 3: M[0, 16]}  {1 -> 0: M[1, 20]}  {4 -> 7: M[4, 12]}  {5 -> 4: M[5, 12]}  {6 -> 5: M[6, 28]}  {7 -> 6: M[7, 32]}  {8 -> 11: M[8, 20]}  {9 -> 8: M[9, 16]}  {10 -> 9: M[10, 32]}  {11 -> 10: M[11, 24]}  {12 -> 15: M[12, 24]}  {13 -> 12: M[13, 32]}  {14 -> 13: M[14, 0]}  {15 -> 14: M[15, 4]}  {16 -> 19: M[16, 28]}  {17 -> 16: M[17, 24]}  {18 -> 17: M[18, 4]}  {19 -> 18: M[19, 8]}  {20 -> 23: M[20, 32]}  {21 -> 20: M[21, 28]}  {22 -> 21: M[22, 8]}  {23 -> 22: M[23, 0]}  {24 -> 27: M[24, 0]}  {25 -> 24: M[25, 8]}  {26 -> 25: M[26, 12]}  {27 -> 26: M[27, 16]}  {28 -> 31: M[28, 4]}  {29 -> 28: M[29, 0]}  {30 -> 29: M[30, 16]}  {31 -> 30: M[31, 20]}  {32 -> 35: M[32, 8]}  {33 -> 32: M[33, 4]}  {34 -> 33: M[34, 20]}  {35 -> 34: M[35, 12]} | Як і передбачається алгоритмом, відбувається передача повідомлень у найбільш відалені вузли. При цьому проходить передача усередині кластера проти годинникової стрілки. Вузол номер 2 не бере участі в передачі: Вузол 3 не передає дані, а 1 – не отримує. У результаті буфери 3-го вузла мітять на одне повідомлення більше за інші, а 1-го – на одне менше. | {1 -> 3: M[1, 16]}  {3 -> 1: M[3, 28]}  {4 -> 6: M[4, 20]}  {5 -> 7: M[5, 16]}  {6 -> 4: M[6, 12]}  {7 -> 5: M[7, 28]}  {8 -> 10: M[8, 12]}  {9 -> 11: M[9, 20]}  {10 -> 8: M[10, 16]}  {11 -> 9: M[11, 32]}  {12 -> 14: M[12, 4]}  {13 -> 15: M[13, 24]}  {14 -> 12: M[14, 8]}  {15 -> 13: M[15, 0]}  {16 -> 18: M[16, 8]}  {17 -> 19: M[17, 28]}  {18 -> 16: M[18, 0]}  {19 -> 17: M[19, 4]}  {20 -> 22: M[20, 0]}  {21 -> 23: M[21, 32]}  {22 -> 20: M[22, 4]}  {23 -> 21: M[23, 8]}  {24 -> 26: M[24, 4]}  {25 -> 27: M[25, 0]}  {26 -> 24: M[26, 8]}  {27 -> 25: M[27, 12]}  {28 -> 30: M[28, 8]}  {29 -> 31: M[29, 4]}  {30 -> 28: M[30, 0]}  {31 -> 29: M[31, 16]}  {32 -> 34: M[32, 0]}  {33 -> 35: M[33, 8]}  {34 -> 32: M[34, 4]}  {35 -> 33: M[35, 20]} | Передача даних продовжується. На даному кроці відбувається передача у «протилежну» вершину. Вузол 0 не бере ніякої участі у даному процесі, оскільки його партнер вийшов з ладу. |

|  |  |
| --- | --- |
| а) | б) |

Рисунок 4.5 Перші кроки маршрутизації

При маршрутизації, вузли, суміжні з 2-гим, відправляють ті пакети, що мали передаватися через нього, на інші вузли.

*Приклад роботи програми*

Відмовили вузли 17 та 33.

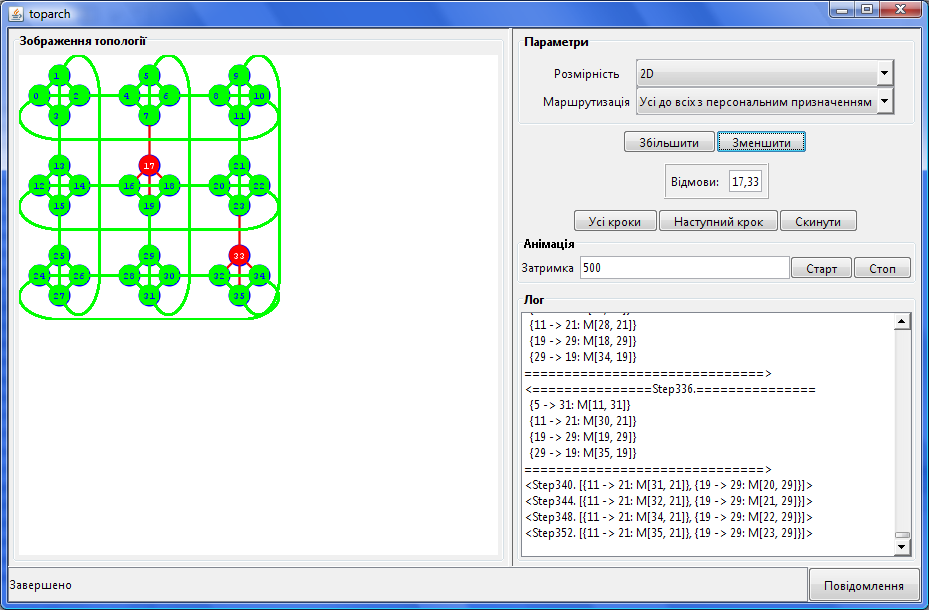


Рисунок 4.6 Робота програми при маршрутизації усі до всіх з персональним призначенням

Розроблений алгоритм розв’язав дану задачу за 352 кроки. Маршрутизувалося 1296 повідомлень.

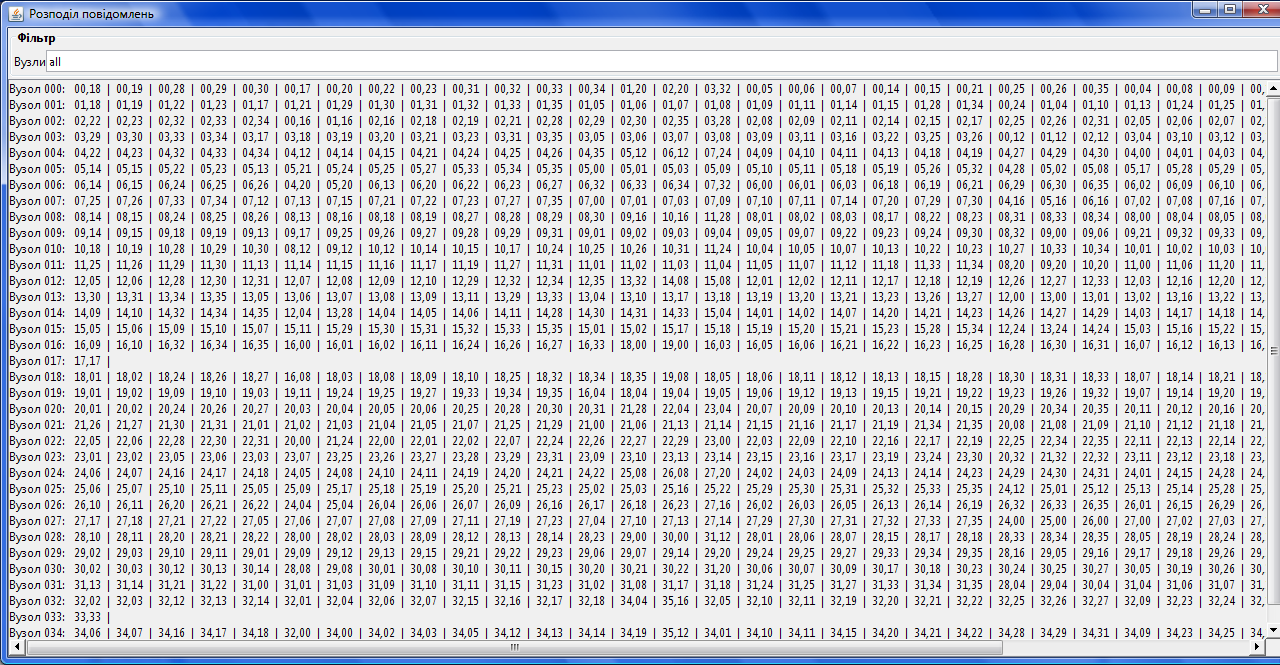


Рисунок 4.7 Розподіл повідомлень після 3 кроку

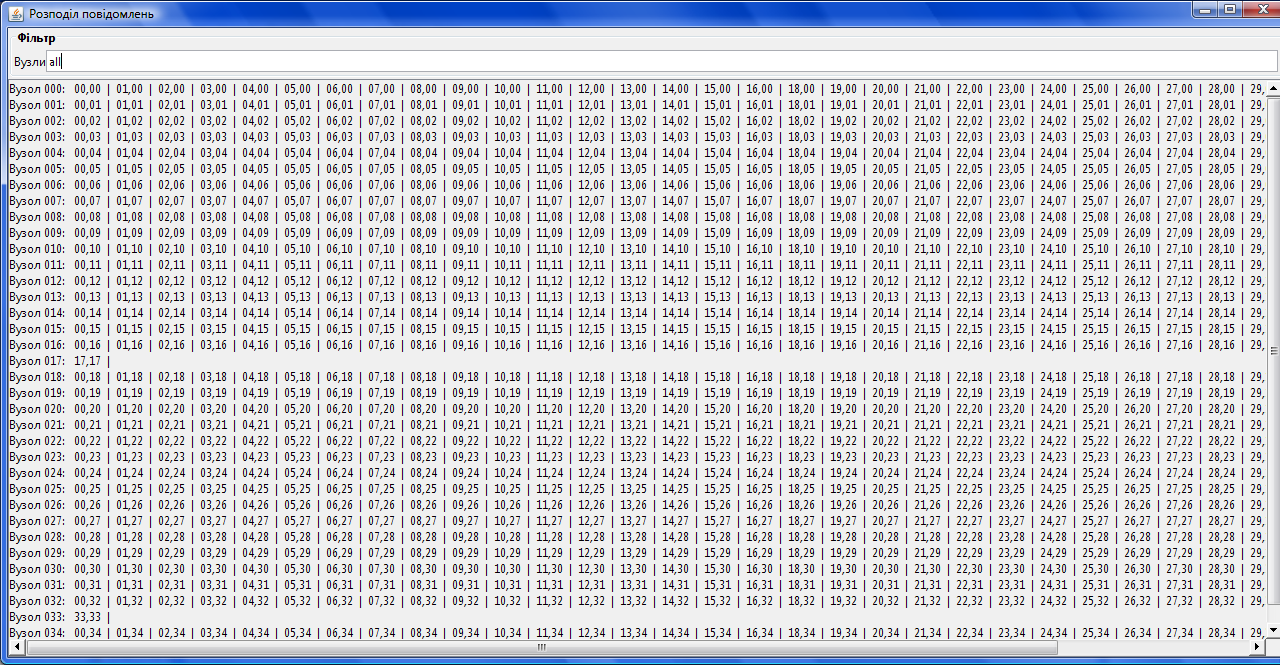


Рисунок 4.8 Розподіл повідомлень після завершення роботи

4.3 Один до всіх

Мета алгоритму – передати одне повідомлення усім вузлам у системі. Така необхідність може виникнути при сповіщенні системи про певні зміни.

Даний алгоритм реалізовано в двох варіантах. Перший, описаний в цьому пункті, основується на лавиноподібній маршрутизації з розчепленням горизонту. Другий оснований на multicast маршрутизації з певними змінами у форматі пакета.

У першому алгоритмі вузол, що містить повідомлення виконує 4 операції передачі даного повідомлення своїм сусідам. При виборі сусіда забороняється відсилати повідомлення вузлу, в якому вже побувало повідомлення. Також виконується перевірка, чи не зайнятий при цьому вузол.

Головним принципом при цьому є передача в першу чергу вузлу, що знаходиться в іншому кластері. У такому випадку на наступному кроці буде задіяно більша кількість кластерів, що скорочує час роботи алгоритму за рахунок одночасних передач.

Приблизний максимальний час маршрутизації складає

Розглянемо приклад маршрутизації при відмові 5 та 6 вузлів (відправник – 3-ій вузол). Лог маршрутизації:

|  |  |
| --- | --- |
| <Step0. [{3 -> 13: Один до всіх}]>  <Step1. [{13 -> 12: Один до всіх}, {3 -> 2: Один до всіх}]>  <===============Step2.===============  {2 -> 4: Один до всіх}  {13 -> 14: Один до всіх}  {12 -> 22: Один до всіх}  {3 -> 0: Один до всіх}  ==============================>  <===============Step3.===============  {2 -> 1: Один до всіх}  {0 -> 10: Один до всіх}  {13 -> 15: Один до всіх}  {22 -> 21: Один до всіх}  {4 -> 7: Один до всіх}  {14 -> 16: Один до всіх}  ==============================> | <===============Step4.===============  {22 -> 23: Один до всіх}  {16 -> 19: Один до всіх}  {12 -> 14: Один до всіх}  {10 -> 9: Один до всіх}  {1 -> 27: Один до всіх}  {14 -> 15: Один до всіх}  {21 -> 11: Один до всіх}  {2 -> 0: Один до всіх}  {15 -> 25: Один до всіх}  {0 -> 1: Один до всіх}  {7 -> 17: Один до всіх}  ==============================>  <===============Step5.===============  {19 -> 29: Один до всіх}  {22 -> 20: Один до всіх}  {16 -> 17: Один до всіх}  {9 -> 35: Один до всіх}  {10 -> 11: Один до всіх}  {17 -> 16: Один до всіх}  {23 -> 33: Один до всіх}  {15 -> 12: Один до всіх}  {25 -> 24: Один до всіх}  {11 -> 10: Один до всіх}  {27 -> 26: Один до всіх}  ==============================> |
| <===============Step6.===============  {35 -> 34: Один до всіх}  {26 -> 28: Один до всіх}  {19 -> 18: Один до всіх}  {9 -> 8: Один до всіх}  {21 -> 23: Один до всіх}  {23 -> 20: Один до всіх}  {25 -> 26: Один до всіх}  {33 -> 32: Один до всіх}  {27 -> 24: Один до всіх}  ==============================>  <===============Step7.===============  {35 -> 32: Один до всіх}  {26 -> 24: Один до всіх}  {19 -> 17: Один до всіх}  {29 -> 30: Один до всіх}  {9 -> 11: Один до всіх}  {17 -> 19: Один до всіх}  {20 -> 21: Один до всіх}  {24 -> 26: Один до всіх}  {25 -> 27: Один до всіх}  {18 -> 20: Один до всіх}  {33 -> 34: Один до всіх}  {11 -> 9: Один до всіх}  {28 -> 31: Один до всіх}  {27 -> 25: Один до всіх}  ==============================> | <===============Step8.===============  {35 -> 33: Один до всіх}  {29 -> 31: Один до всіх}  {8 -> 10: Один до всіх}  {30 -> 32: Один до всіх}  {31 -> 30: Один до всіх}  {32 -> 34: Один до всіх}  {18 -> 17: Один до всіх}  {33 -> 35: Один до всіх}  {28 -> 29: Один до всіх}  ==============================>  <===============Step9.===============  {30 -> 28: Один до всіх}  {18 -> 16: Один до всіх}  {28 -> 30: Один до всіх}  ==============================> |

На рисунку нижче показано передачі повідомлень на 3-ому кроці.

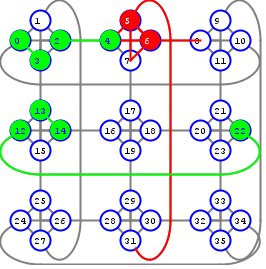


Рисунок 4.9 Маршрутизація один до всіх (лавиноподібний варіант)

*Приклад роботи програми*

Відправник – вузол №4. Відмовили 6 та 31.

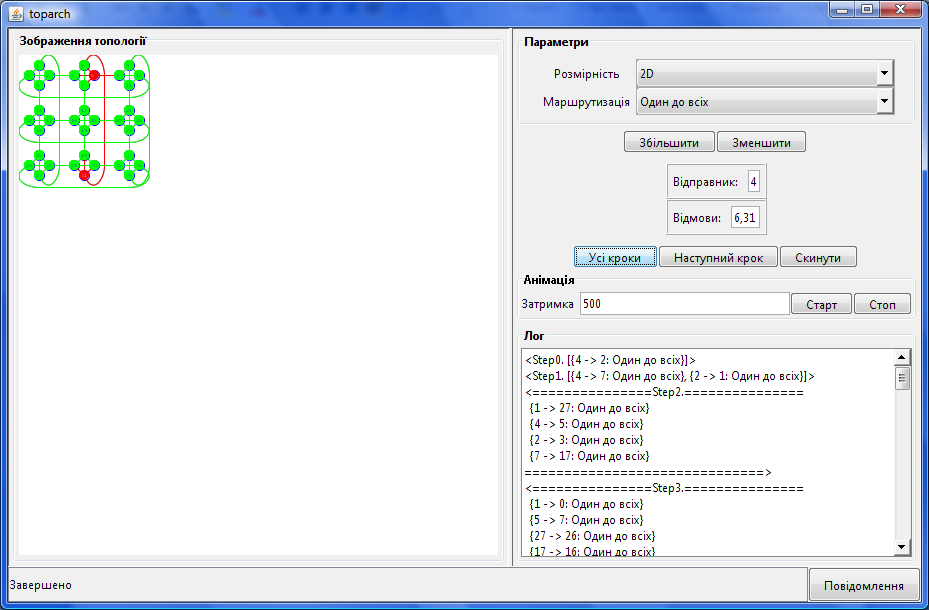


Рисунок 4.10 Робота програми при маршрутизації один до всіх

4.4 Multicast

Даний тип маршрутизації передбачає відправку даних групі вузлів у системі і може бути використаним у багатьох задачах, що розв’язуються в системі.

Для даної маршрутизації у кожного вузла має бути присутній буфер повідомлень. Для узагальнення, розглядатимемо початок роботи алгоритму як прихід отримання вузлом-відправником повідомлення, що має бути розіслане певній групі отримувачів. Адреси отримувачів містяться у заголовку отриманого пакета.

При отриманні повідомлення відбувається аналіз адрес кінцевих вузлів. Для кожного з них знаходиться знаходиться наступний вузол, куди буде здійснено передачу за принципом алгоритму «один до одного». Після цього кінцеві вузли групуються відповідно до наступного вузла. Не важко здогадатися, що кількість таких груп не перевищуватиме 4. Далі, для кожної з груп формується нове повідомлення, в поле адреси приймачів записуються кінцеві вузли, і кожне таке повідомлення додається у вихідний буфер.

Якщо вихідний буфер вузла не пустий, то з нього вибирається перше повідомлення, для якого наступний вузол не зайнятий (відповідний лінк перебуває у стані готовності). Коли таке повідомлення вибране, відбувається передача по каналу, повідомлення видаляється з вихідного буферу.

Потрібно відзначити, що вихідний буфер, так само як і у варіанті «всі до всіх з персональним призначення» підтримує пріоритетну чергу для того, щоб у першу чергу відправляти повідомлення з найвіддаленішими кінцевими вузлами.

Розглянемо приклад маршрутизації. Відправник – вузол 0. Отримувачі: 5, 10, 11, 35, 33. Відмовив вузол 2.

Лог маршрутизації:

<Step1. [{0 -> 10: [5, 10, 11, 35, 33]}]>

<Step2. [{10 -> 8: [5]}]>

<Step3. [{10 -> 9: [35, 33]}, {8 -> 6: [5]}]>

<===============Step4.===============

{9 -> 35: [35, 33]}

{6 -> 5: [5]}

{10 -> 11: [11]}

==============================>

<Step5. [{35 -> 33: [33]}]>

Задача була розв’язана за 5 кроків. На рисунку нижче наведено дерево роботи multicast маршрутизації.

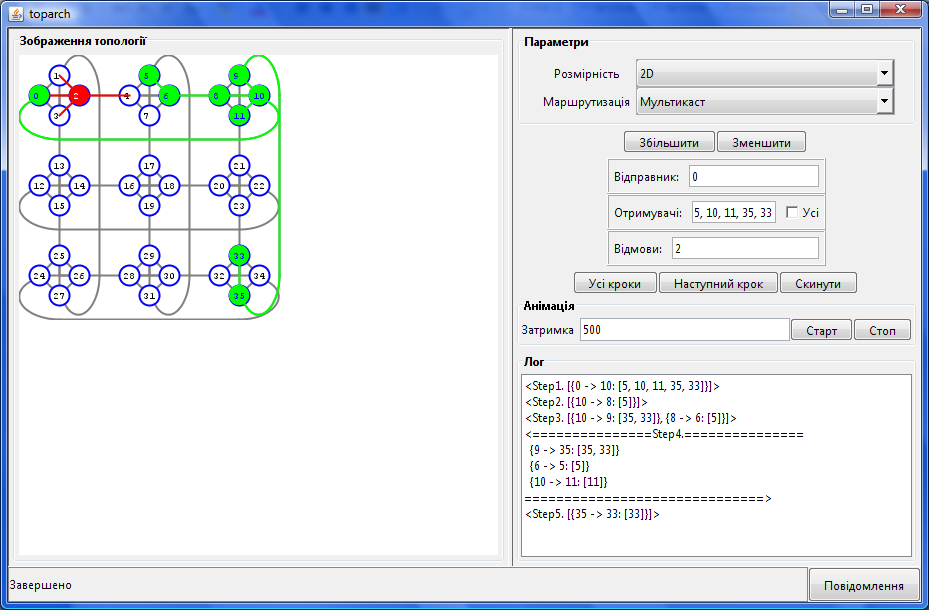
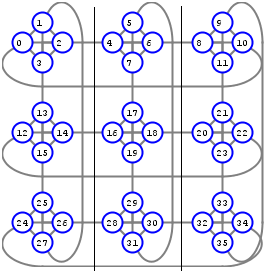


Рисунок 4.11 Multicast маршрутизація

Важливо відзначити, що дана маршрутизація може бути ще одним варіантом маршрутизації один до всіх. При цьому, для скорочення поля адрес приймачів доцільно використовувати маски. Наприклад, при передачі з вузла №2 повідомлень усім вершинам, що мають одиничку в першій координаті кластера (рисунок 4.12), їхні адреси можна закодувати двома числами – 1 (формат з маскою), 1 (адреса). Таким чином, отримаємо значне скорочення розміру пакетів та ефективніший трафік.



**21**

**11**

**01**

Рисунок 4.12 Маскування адрес приймачів

5 Завантаження прикладної задачі

5.1 Генетичні алгоритми в еволюції

В області науки еволюції існує задача моделювання розвитку біологічних видів. Часто використовується генетичний алгоритм для її розв’язання. У ході своєї роботи алгоритм генерує множини поколінь, для яких часто виникає потреба моделювання їхнього процесу життєдіяльності.

Генетичний алгоритм включає використання сукупності розв’язків, які отримують зі значень природного добору. Сукупність побудована із вирівнювань. Ініціюючим кроком є створення нульового покоління , що створюється стохастично. Величина сукупності зберігається постійною сталою. З переходом від одного покоління до наступного діти беруться від батьків, які обрані природним добором, що основується на їх пристосованості (чим краще пристосованість, тим більше дітей). Для створення нащадка оператором може бути обраний кросинговер або мутація. Ці три кроки постійно повторюються. Впродовж циклів нові рядки вирівнювання з’являються через мутації, які комбінуються кросинговером. Вибірка задовольняє умову, що буде виживати краща частина, а динамічна зміна оператора допомогає сукупності в створенні нащадків, яких вона потребує. Пристосованість сукупності зростає, поки жодне покращення вже не може бути зроблено.

Генетичний алгоритм можна розбити на такі етапи:

1. Ініціалізація:
   1. Створення ;
2. Визначення, оцінка:
   1. Оцінити сукупність *n*-ої генерації.
   2. Якщо популяція стабільне, перейти в кінець.
   3. Вибір індивідуумів для переміщення.
   4. Оцінити очікуване потомство.
3. Розмноження:
   1. Обрати батьків із сукупності,
   2. Обрати оператор,
   3. Згенерувати нових дітей,
   4. Прийняти чи відмовитися від нового індивідуума для покоління ,
   5. Перейти на f, поки сукупність не буде сформована,
   6. Прийняти ,
   7. Перейти до оцінки
4. Кінець

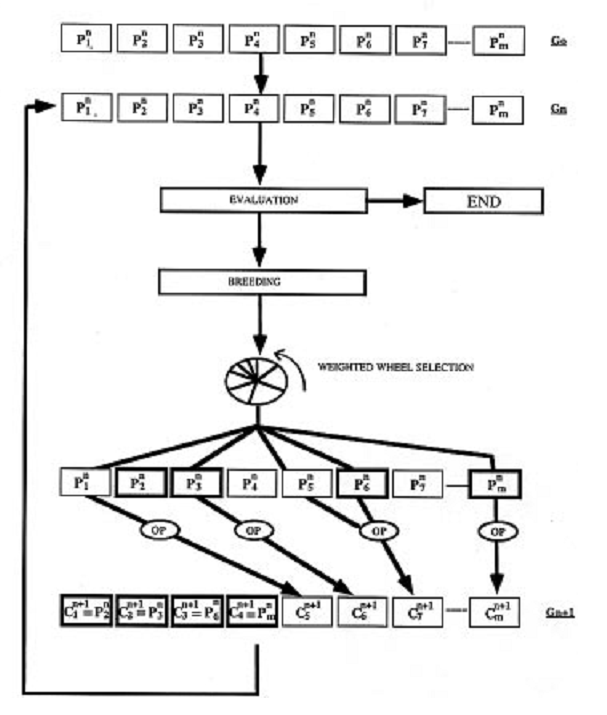


Рисунок 5.1 Схематичне зображення генетичного алгоритму

Перший крок алгоритму – це створення випадкової сукупності. Складається з ряду вирівнювань, які містять тільки термінальні пропуски. Для створення одного з вирівнювань обирається нащадок для всіх послідовностей (від 0 до 50 основ, якщо послідовність наприклад 200 основ) і всі послідовності пересовуються вправо, згідно із нащадком.

Наступним кроком є оцінка. Дає народження новій генерації. Для цього оцінюється пристосованість кожного індивідуума. Ця пристосованість оцінюється через очки кожного вирівнювання згідно із задачею оптимізації. Очікувані нащадки із вирівнювання отримують з пристосованості. Частина популяції пересувається в наступне покоління (близько 50 %). Тобто сукупність перекрита, половина вирівнювання лишається без змін, а інша замінена на нащадки.

Розмноження. Нове покоління заповнюється найкращими індивідуумами з попередньої сукупності (50%). Інша частина заповнюється через створення нових нащадків з відібраних батьків. Впродовж цього кроку очікувані нащадки (залежить від пристосованості) використовується як ймовірність для кожного індивіда бути обраним як батько. Для перетворення батьків має бути обраний оператор. Оператор це невелика програма, яка перетворює вирівнювання (зміщує пропуски, з двох вирівнювань робить одне).

В розмноженні використовується досить важливий критерій – оператор. В основному він виконує дві задачі – кросинговер і мутації. Виконує він такі задачі:

* Кросинговер (об’днання двох вирівнювань в одне),
* Вставка пропуску,
* Перемішування блоків (коротка частина вирівнювань без пропусків),
* Локальна оптимальна чи субоптимальна перетасовка.

5.2 Завантаження у топологію

Послідовності нуклеотидів, над якими виконуються дії генетичних операторів можуть бути закодовані трісковою системою числення. У такому випадку ланцюжок, що представляє певне покоління, є нічим іншим, як номер кластера, на якому буде проводитися його моделювання.

При цьому мутація у ланцюжкові спричинить перехід до сусіднього кластера.



Рисунок 5.2 Робота на синтезованій топологічній організації

Породження нового покоління не означає припинення моделювання поточного. Таким чином отримуємо паралельну роботу при моделюванні різних еволюційних поколінь.

При операціях кросинговеру будуть використовуватися алгоритми маршрутизації, описані вище.

Висновки

Результатом виконання курсового проекту є синтезована топологічна організація. Вона має сталий показник степеня та властивості трійкового гіперкуба. Діаметр зростає за логарифмічним законом. Топологія має меншу вартість за гіперкуб, саме за рахунок сталого степеню. Синтезована топологія була відображена у класичний гіперкуб.

Для даної організації були розроблені алгоритми маршрутизації з урахуванням вимоги відмовостійкості: один до одного, всі до всіх з персональним призначенням, один до всіх та multicast. Алгоритми розв’язують проблеми з відмовами у динаміці, вносячи збитковість у маршрут даних, що передаються.

У якості прикладу використання топології було розглянуто задачу моделювання поколінь біологічного виду за допомогою генетичного алгоритму, що використовується в еволюції.

Загалом, розроблена система може використовуватися за умови, коли показники степеня та вартості посідають головну роль, або необхідна достатньо надійна система з властивостями гіперкубічної організації.

Окрім того, було розроблено програмний продукт, на якому проводилося моделювання роботи системи. Він дозволяє проаналізувати роботу реалізваних алгоритмів та побудувати план завантаження користувацької задачі в систему.

Додаток

Лістинг програми

package org.mazur.toparch;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Comparator;

import java.util.LinkedList;

import java.util.List;

public class Utils {

private static RouteFilter ALL\_WORKING = new RouteFilter() {

@Override

public boolean accept(int nextNode) { return true; }

};

/\*\*

\* We have 2 connectors for the same axis: 0 and 1.

\* ---------

\* 0 1

\* x1-0x1-0x

\*/

private static byte[][] CONNECTORS = new byte[][] {

{-1, 1, 0},

{ 0, -1, 1},

{ 1, 0, -1}

};

public static boolean isSameCluster(final int i, final int j, int d) {

int cs = d << 1;

return i / cs == j / cs;

}

/\*\*

\* @param i source cluster

\* @param j dest cluster

\* @param d dimension

\* @return array of different axises

\*/

public static int[] compareClusters(final int i, final int j, final int d) {

if (i == j) { return null; }

int dc = 0;

int[] tempResult = new int[d];

int a = i > j ? i : j;

int b = i > j ? j : i;

int axis = 0;

while (a > 0) {

int d1 = a % 3, d2 = b % 3;

if (d1 != d2) {

tempResult[dc] = axis;

dc++;

}

a /= 3; b /= 3;

axis++;

}

int[] result = new int[dc];

System.arraycopy(tempResult, 0, result, 0, dc);

return result;

}

/\*\*

\* @param i source cluster

\* @param j destination cluster

\* @param dimension

\* @return near info

\*/

public static NearInfo getNearInfo(final int i, final int j, final int d) {

int a = i > j ? i : j;

int b = i > j ? j : i;

int dc = 0;

int axisIndex = 0, axis = -1;

int rd1 = 0, rd2 = 0;

while (a > 0) {

int d1 = a % 3, d2 = b % 3;

if (d1 != d2) { dc++; }

if (dc > 1) { return null; }

if (d1 != d2) {

axis = axisIndex;

rd1 = i > j ? d1 : d2;

rd2 = i > j ? d2 : d1;

}

a /= 3; b /= 3;

axisIndex++;

}

int source = d \* CONNECTORS[rd1][rd2] + axis;

int destination = d \* CONNECTORS[rd2][rd1] + axis;

return new NearInfo(axis, source, destination);

}

public static boolean isNear(final int i, final int j) {

return getNearInfo(i, j, 0) != null;

}

/\*\*

\* @return true if nodes are connected

\*/

public static boolean isConnected(final int i, final int j, final int k) {

final int clusterSize = k << 1;

int ci = i / clusterSize, cj = j / clusterSize;

if (ci == cj) {

int d = Math.abs(i - j);

return d == 1 || d == clusterSize - 1 || d == k;

}

NearInfo info = getNearInfo(ci, cj, k);

if (info == null) { return false; }

return info.source == i % clusterSize && info.dest == j % clusterSize;

}

public static int getDigit(int number, int d) {

while(d-- > 0) { number /= 3; }

return number % 3;

}

public static int setDigit(int number, final int d, final int v) {

// save lower digits

int[] saved = new int[d];

for (int i = 0; i < d; i++) {

saved[i] = number % 3;

number /= 3;

}

// correction

number = (number / 3) \* 3 + v;

// restore lower digits

for (int i = d - 1; i >= 0; i--) {

number \*= 3;

number += saved[i];

}

return number;

}

/\*\*

\* @param node node number

\* @param d dimension

\* @return connected node in the other cluster

\*/

public static int getNearClusterConnection(final int node, final int d) {

int cs = d << 1;

int ci = node / cs;

int ni = node % cs;

int axis = ni % d;

int sourceConnector = ni / d;

int nearDigit = getDigit(ci, axis);

nearDigit += sourceConnector == 0 ? -1 : 1;

nearDigit %= 3;

if (nearDigit < 0) { nearDigit = 3 + nearDigit; }

int nearCluster = setDigit(ci, axis, nearDigit);

return nearCluster \* cs + (ni + d) % cs;

}

/\*\*

\* @param node node

\* @return array of connected nodes

\*/

public static Integer[] getConnected(final int node, final int d) {

int cs = d << 1;

int startNode = node / cs \* cs;

return new Integer[] {

startNode + (node + 1) % cs,

startNode + (node > 0 ? (node - 1) % cs : cs - 1),

startNode + (node + d) % cs,

getNearClusterConnection(node, d)

};

}

public static int getInClusterDistance(final int i, final int j, final int d) {

int next = getNextInCluster(i, j, d);

if (next == i) { return 0; }

try {

return getInClusterDistance(next, j, d) + 1;

} catch (Throwable e) {

throw new RuntimeException(e);

}

}

public static int getNextRouteInCluster(final int i, final int j, final int d, final RouteFilter filter) {

int cs = d << 1;

// form partners

List<Integer> partners = new ArrayList<Integer>(Arrays.asList((i + 1) % cs, (i + d) % cs, i == 0 ? cs - 1 : i - 1));

// sort them to define those that give the minimum distance

Collections.sort(partners, new Comparator<Integer>() {

@Override

public int compare(Integer p1, Integer p2) { return getInClusterDistance(p1, j, d) - getInClusterDistance(p2, j, d); }

});

int next = -1;

// find

for (int p : partners) {

if (filter.accept(p)) { next = p; break; }

}

return next;

}

public static int getNextInCluster(final int i, int j, final int d) {

int cs = d << 1;

int dif = j - i;

if (dif == 0) { return i; }

int sign = dif > 0 ? 1 : -1;

dif = Math.abs(dif);

int next = 0;

if (dif > d) {

next = (i + (dif < d + (d >> 1) ? d : -sign)) % cs;

} else {

next = (i + (dif > d >> 1 ? d : sign)) % cs;

}

if (next < 0) { next = cs + next; }

return next;

}

public static int getNodesCount(final int d) {

return ((int)(Math.pow(3, d)) \* d) << 1;

}

public static int getNextNode(final int i, final int j, final int d, final RouteFilter filter) {

if (i == j) { return -1; }

final int cs = d << 1;

final int ci = i / cs, cj = j /cs;

final int clusterSource = i % cs;

RouteFilter filterWrapper = new RouteFilter() {

@Override

public boolean accept(final int clusterIndex) { return filter.accept(ci \* cs + clusterIndex); }

};

if (ci == cj) { // same cluster

int clusterDest = j % cs;

int res = getNextRouteInCluster(clusterSource, clusterDest, d, filterWrapper);

if (res < 0) { return -1; }

return ci \* cs + res;

} else { // between clusters

// find transition clusters

int[] axises = compareClusters(ci, cj, d);

int[] transitionClusters = new int[axises.length];

for (int k = 0; k < transitionClusters.length; k++) {

transitionClusters[k] = Utils.setDigit(ci, axises[k], Utils.getDigit(cj, axises[k]));

}

// define jump points to that clusters

List<Integer> jumpPoints = new ArrayList<Integer>(cs);

for (int cluster : transitionClusters) {

jumpPoints.add(Utils.getNearInfo(ci, cluster, d).getSource());

}

Collections.sort(jumpPoints, new Comparator<Integer>() {

@Override

public int compare(Integer p1, Integer p2) {

return Utils.getInClusterDistance(p1, clusterSource, d) - Utils.getInClusterDistance(p2, clusterSource, d);

}

});

List<Integer> altJP = new LinkedList<Integer>();

for (Integer jp : jumpPoints) { altJP.add((jp + d) % cs); }

jumpPoints.addAll(altJP);

for (int k = 0; k < cs; k++) { if (!jumpPoints.contains(k)) { jumpPoints.add(k); } }

// try jump points

for (Integer njp : jumpPoints) {

int jpNode = cs \* ci + njp;

int nextNode = getNearClusterConnection(jpNode, d);

if (!filter.accept(nextNode)) { continue; } // we were there

if (i == jpNode) {

// jump

return nextNode;

} else {

// move in cluster[

if (!filter.accept(jpNode)) { continue; }

int res = getNextRouteInCluster(clusterSource, njp, d, filterWrapper);

if (res >= 0) { return ci \* cs + res; }

}

}

return -1;

}

}

public static int getNextNode(int i, int j, int d) { return getNextNode(i, j, d, ALL\_WORKING); }

public static int getRouteDistance(final int i, final int j, final int d) {

int currentNode = i;

int res = 0;

while (currentNode != j) {

currentNode = getNextNode(currentNode, j, d);

res++;

}

return res;

}

private Utils() { /\* hidden \*/ }

public static class NearInfo {

private int axis, source, dest;

public NearInfo(final int axis, final int source, final int dest) {

this.axis = axis;

this.source = source;

this.dest = dest;

}

/\*\*

\* @return the axis

\*/

public int getAxis() { return axis; }

/\*\*

\* @return the source

\*/

public int getSource() { return source; }

/\*\*

\* @return the dest

\*/

public int getDest() { return dest; }

@Override

public String toString() {

return "Near[axis = " + axis + ", " + source + " <-> " + dest + "]";

}

}

public static void main(final String[] args) {

System.out.println(getNearInfo(0, 3, 3));

}

}

package org.mazur.toparch.model;

import java.util.Comparator;

import java.util.List;

import java.util.TreeSet;

import org.mazur.toparch.RouteFilter;

import org.mazur.toparch.State;

import org.mazur.toparch.Utils;

import org.mazur.toparch.router.LinkDescriptor;

/\*\*

\* Node containing messages.

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

public class Node {

private int number;

private Comparator<Message> mComparator = new Comparator<Message>() {

@Override

public int compare(final Message m1, final Message m2) {

if (m1.equals(m2)) { return 0; }

int d = State.INSTANCE.getDimension();

int d1 = Utils.getRouteDistance(number, m1.getDestination(), d);

int d2 = Utils.getRouteDistance(number, m2.getDestination(), d);

int dif = d2 - d1;

if (dif != 0) { return dif; }

return 1000 \* (m1.getSource() - m2.getSource()) + (m1.getDestination() - m2.getDestination());

}

};

private TreeSet<Message> buffers = new TreeSet<Message>(mComparator);

/\*\*

\* @return the number

\*/

public int getNumber() { return number; }

public static String formNumber(final int number, final int length) {

String res = String.valueOf(number);

if (res.length() >= length) { return res; }

StringBuilder sb = new StringBuilder(res);

int l = length - res.length();

char[] zeros = new char[l];

for (int i = 0; i < l; i++) { zeros[i] = '0'; }

sb.insert(0, zeros);

return sb.toString();

}

protected String describeMessage(final Message m, final int zerosCount) {

return formNumber(m.getSource(), zerosCount) + "," + formNumber(m.getDestination(), zerosCount);

}

/\*\*

\* @return the messages

\*/

public String[] getMessages() {

String[] result = new String[buffers.size()];

int index = 0;

int d = State.INSTANCE.getDimension();

int n = Utils.getNodesCount(d);

int zerosCount = 0;

while (n > 0) { zerosCount++; n /= 10; }

for (Message m : buffers) {

result[index++] = describeMessage(m, zerosCount);

}

return result;

}

/\*\*

\* @param number the number to set

\*/

public void setNumber(int number) { this.number = number; }

/\*\*

\* @param messages the messages to set

\*/

public void addMessage(final Message m) {

assert m != null : "Attempt to add null message to node " + this;

this.buffers.add(m.copy());

}

public Message removeMessage(final int from, final int to) {

Message m = new Message(from, to, null);

return removeMessage(m);

}

public Message removeMessage(final Message m) {

Message result = buffers.floor(m);

assert result != null;

this.buffers.remove(result);

return result;

}

public void copyMessages(final Node node) {

this.buffers = new TreeSet<Message>(node.buffers);

}

protected static boolean isKilled(final int src, final int dst, final List<LinkDescriptor> killed) {

if (killed == null || killed.isEmpty()) { return false; }

for (LinkDescriptor ld : killed) {

if ((ld.getSource() == src && ld.getDestination() == dst)

|| (ld.getSource() == dst && ld.getDestination() == src)) {

return true;

}

}

return false;

}

public static RouteFilter createRouteFilter(final int number, final Message m, final List<LinkDescriptor> killed) {

return new RouteFilter() {

@Override

public boolean accept(int nextNode) {

return !m.getVisitedNodes().get(nextNode) && !isKilled(number, nextNode, killed);

}

};

}

public Message selectMessage(final int nextNode, final List<LinkDescriptor> killed, final List<Message> removed) {

Message result = null;

int d = State.INSTANCE.getDimension();

for (final Message m : buffers) {

int mNext = Utils.getNextNode(number, m.getDestination(), d, createRouteFilter(number, m, killed));

if (mNext == -1 && m.getDestination() != number) {

removed.add(m);

}

if (nextNode == mNext) { result = m; break; }

}

return result;

}

protected TreeSet<Message> getBuffers() {

return buffers;

}

@Override

public String toString() {

return "N" + number + "{buffersSize:" + buffers.size() + "}";

}

}

package org.mazur.toparch.router;

import java.util.Collections;

import java.util.LinkedList;

import java.util.Set;

import javax.swing.JPanel;

import org.mazur.toparch.State;

import org.mazur.toparch.model.Node;

import org.mazur.toparch.play.PlayList;

import org.mazur.toparch.play.StepInfo;

import org.mazur.toparch.Utils;

/\*\*

\* Router that does a model process.

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

public abstract class Router<T extends InputData> {

/\*\* Factory. \*/

private InputDataPanelFactory<T> inputDataFactory;

protected static HopResolver[] STANDARD\_RESOLVERS = new HopResolver[] {

// =================== circle routing ===================

new HopResolver() {

@Override

public int getNext(final int current) {

int cs = State.INSTANCE.getDimension() << 1;

int cluster = current / cs;

int ci = current % cs;

ci--;

if (ci < 0) { ci += cs; }

return cluster \* cs + ci;

}

},

// =================== opposite routing ===================

new HopResolver() {

@Override

public int getNext(final int current) {

int d = State.INSTANCE.getDimension();

int cs = d << 1;

int cluster = current / cs;

int ci = current % cs;

ci += d; ci %= cs;

return cluster \* cs + ci;

}

},

// =================== circle routing ===================

new HopResolver() {

@Override

public int getNext(final int current) {

int cs = State.INSTANCE.getDimension() << 1;

int cluster = current / cs;

int ci = (current % cs + 1) % cs;

return cluster \* cs + ci;

}

},

// =================== clusters routing ===================

new HopResolver() {

@Override

public int getNext(final int current) {

return Utils.getNearClusterConnection(current, State.INSTANCE.getDimension());

}

}

};

public Router() {

inputDataFactory = createFactory();

}

/\*\*

\* @return name

\*/

public abstract String getName();

/\*\*

\* @return GUI panel

\*/

public JPanel getGUIPanel() { return inputDataFactory.getPanel(); }

protected abstract InputDataPanelFactory<T> createFactory();

public Set<? extends Node> getMarkedNodes() { return Collections.emptySet(); }

/\*\*

\* Prepare.

\*/

public abstract void reinit();

/\*\*

\* @return the next step info

\*/

public StepInfo next() { return next(formData()); }

protected abstract StepInfo next(final T input);

protected T formData() { return inputDataFactory.formData(); }

/\*\*

\* @return play list

\*/

public PlayList process() {

reinit();

//T inputData = formData();

PlayList result = new PlayList();

result.setName("Full play list for " + getName());

result.setStepsInfo(new LinkedList<StepInfo>());

while (true) {

StepInfo si = next();

if (si == null) { break; }

result.getStepsInfo().add(si);

}

return result;

}

protected InputDataPanelFactory<T> getInputDataFactory() {

return inputDataFactory;

}

}

package org.mazur.toparch.router.all2all.personolized;

import static org.mazur.toparch.Utils.getNodesCount;

import java.util.ArrayList;

import java.util.BitSet;

import java.util.LinkedList;

import java.util.List;

import org.mazur.toparch.State;

import org.mazur.toparch.model.Message;

import org.mazur.toparch.model.Node;

import org.mazur.toparch.play.HopInfo;

import org.mazur.toparch.play.StepInfo;

import org.mazur.toparch.router.HopResolver;

import org.mazur.toparch.router.InputDataPanelFactory;

import org.mazur.toparch.router.LinkDescriptor;

import org.mazur.toparch.router.Router;

import org.mazur.toparch.router.all2all.A2AInputsFactory;

import org.mazur.toparch.router.all2all.A2ARouterInputs;

/\*\*

\* All to all personalized router.

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

public class A2APRouter extends Router<A2ARouterInputs> {

private List<Node> nodes, nextStepNodes;

private int step = 0;

@Override

protected InputDataPanelFactory<A2ARouterInputs> createFactory() { return new A2AInputsFactory(); }

@Override

public String getName() { return "Усі до всіх з персональним призначенням"; }

private void send(Message m, final int from, final int to) {

Node sourceNode = nodes.get(from);

sourceNode.removeMessage(m);

sourceNode = nextStepNodes.get(from);

Node destinationNode = nextStepNodes.get(to);

m = sourceNode.removeMessage(m);

m.getVisitedNodes().set(from);

destinationNode.addMessage(m);

}

public String[][] formMDistrib() {

String[][] result = new String[nextStepNodes.size()][];

int index = 0;

for (Node node : nextStepNodes) {

result[index++] = node.getMessages();

}

return result;

}

private void copyNodes() {

if (nextStepNodes != null) { nodes = nextStepNodes; }

nextStepNodes = new ArrayList<Node>(nodes.size());

for (Node node : nodes) {

Node copy = new Node();

copy.setNumber(node.getNumber());

copy.copyMessages(node);

nextStepNodes.add(copy);

}

}

@Override

public void reinit() {

int d = State.INSTANCE.getDimension();

int n = getNodesCount(d);

nodes = new ArrayList<Node>(n);

for (int i = 0; i < n; i++) {

Node node = new Node();

node.setNumber(i);

for (int j = 0; j < n; j++) { node.addMessage(new Message(i, j, new BitSet(n))); }

nodes.add(node);

}

step = 0;

nextStepNodes = null;

}

private HopInfo resolve(final Node node, final HopResolver resolver, final List<LinkDescriptor> killed) {

int nextNode = resolver.getNext(node.getNumber());

LinkedList<Message> removed = new LinkedList<Message>();

Message message = node.selectMessage(nextNode, killed, removed);

for (Message toRemove : removed) {

node.removeMessage(toRemove);

nextStepNodes.get(node.getNumber()).removeMessage(toRemove);

}

if (message == null) { return null; }

StringBuilder description = new StringBuilder();

send(message, node.getNumber(), nextNode);

description.append(message);

HopInfo info = new HopInfo();

info.setDescription(description.toString());

info.setSource(node.getNumber());

info.setDestination(nextNode);

return info;

}

private List<HopInfo> runResolver(final HopResolver resolver, final List<LinkDescriptor> killed) {

List<HopInfo> hops = new LinkedList<HopInfo>();

for (Node node : nodes) {

HopInfo hi = resolve(node, resolver, killed);

if (hi != null) { hops.add(hi); }

}

return hops;

}

@Override

protected StepInfo next(final A2ARouterInputs input) {

System.out.println("Step " + step);

copyNodes();

StepInfo result = new StepInfo();

int internalStep = 0;

while (internalStep < STANDARD\_RESOLVERS.length) {

HopResolver resolver = STANDARD\_RESOLVERS[step++ % STANDARD\_RESOLVERS.length];

List<HopInfo> hops = runResolver(resolver, input.getKilled());

if (hops.isEmpty()) {

internalStep++;

continue;

}

result.setHopsInfo(hops);

break;

}

if (result.getHopsInfo() == null) { return null; }

result.setStep(step);

result.setMessagesDistribution(formMDistrib());

return result;

}

}

package org.mazur.toparch.router.multicast;

import java.util.ArrayList;

import java.util.BitSet;

import java.util.HashMap;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.Map.Entry;

import org.mazur.toparch.State;

import org.mazur.toparch.Utils;

import org.mazur.toparch.model.Message;

import org.mazur.toparch.model.Node;

import org.mazur.toparch.play.HopInfo;

import org.mazur.toparch.play.StepInfo;

import org.mazur.toparch.router.InputDataPanelFactory;

import org.mazur.toparch.router.LinkDescriptor;

import org.mazur.toparch.router.Router;

/\*\*

\*

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

public class MulticastRouter extends Router<MulticastInputs> {

/\*\* Senders. \*/

private HashSet<GenericNode> senders;

/\*\* Step. \*/

private int step = 0, d;

/\*\* Nodes. \*/

private List<GenericNode> nodes;

private List<LinkDescriptor> killed;

/\*\* Busy nodes. \*/

private BitSet busyNodes;

/\*\* Marked nodes. \*/

private Set<GenericNode> markedNodes;

@Override

protected InputDataPanelFactory<MulticastInputs> createFactory() {

return new MulticastInputsFactory();

}

@Override

public String getName() { return "Мультикаст"; }

private void addSender(final int number, final List<Integer> destinations) {

GenericNode node = nodes.get(number);

GenericMessage msg = new GenericMessage(number, destinations, new BitSet(Utils.getNodesCount(d)));

node.addMessage(msg);

senders.add(node);

}

@Override

public Set<? extends Node> getMarkedNodes() { return markedNodes; }

@Override

protected StepInfo next(final MulticastInputs input) {

step++;

busyNodes.clear();

killed = input.getKilled();

if (senders == null) {

senders = new HashSet<GenericNode>();

addSender(input.getSource(), input.getDestinations());

}

StepInfo result = new StepInfo();

result.setStep(step);

List<HopInfo> hops = new LinkedList<HopInfo>();

for (GenericNode s : new ArrayList<GenericNode>(senders)) {

if (s.getMessages().length == 0) { senders.remove(s); }

GenericMessage m = s.nextMessage();

if (m == null) { continue; }

HopInfo hop = s.send(m);

if (hop == null) { continue; }

hops.add(hop);

}

if (!hops.isEmpty()) { result.setHopsInfo(hops); }

markedNodes.addAll(senders);

if (result.getHopsInfo() == null) { return null; }

return result;

}

public String[][] formMDistrib() {

String[][] result = new String[nodes.size()][];

int index = 0;

for (Node node : nodes) {

result[index++] = node.getMessages();

}

return result;

}

@Override

public void reinit() {

d = State.INSTANCE.getDimension();

step = 0;

senders = null;

int n = Utils.getNodesCount(d);

busyNodes = new BitSet(n);

nodes = new ArrayList<GenericNode>(n);

for (int i = 0; i < n; i++) {

GenericNode node = new GenericNode();

node.setNumber(i);

nodes.add(node);

}

markedNodes = new HashSet<GenericNode>(n);

}

/\*\*

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

private class GenericNode extends Node {

@Override

public void addMessage(final Message m) {

GenericMessage msg = (GenericMessage)m;

Map<Integer, List<Integer>> mGroups = new HashMap<Integer, List<Integer>>();

for (int dest : msg.destinations) {

int next = Utils.getNextNode(getNumber(), dest, d, createRouteFilter(getNumber(), msg, killed));

List<Integer> group = mGroups.get(next);

if (group == null) { group = new ArrayList<Integer>(); mGroups.put(next, group); }

group.add(dest);

}

for (Entry<Integer, List<Integer>> e : mGroups.entrySet()) {

boolean exist = false;

for (Message existedM : getBuffers()) {

if (((GenericMessage)existedM).getPartner(getNumber()) == e.getKey()) {

((GenericMessage)existedM).destinations.addAll(e.getValue());

exist = true;

break;

}

}

if (exist) { continue; }

BitSet bs = new BitSet();

bs.or(m.getVisitedNodes());

GenericMessage nm = new GenericMessage(m.getSource(), e.getValue(), bs);

nm.setDestination(e.getValue().get(0));

nm.partner = e.getKey();

super.addMessage(nm);

}

}

public GenericMessage nextMessage() {

for (Message m : getBuffers()) {

int next = Utils.getNextNode(getNumber(), m.getDestination(), d, createRouteFilter(getNumber(), m, killed));

if (next == -1) { continue; }

if (!busyNodes.get(next)) { return (GenericMessage)m; }

}

return null;

}

public HopInfo send(final GenericMessage msg) {

int partner = msg.getPartner(getNumber());

if (partner < 0) { return null; }

removeMessage(msg);

msg.getVisitedNodes().set(getNumber());

GenericNode dest = nodes.get(partner);

dest.addMessage(msg);

senders.add(dest);

busyNodes.set(partner);

HopInfo hop = new HopInfo();

hop.setSource(getNumber());

hop.setDestination(partner);

hop.setDescription(msg.destinations.toString());

return hop;

}

@Override

protected String describeMessage(Message m, int zerosCount) {

return m.getSource() + "," + ((GenericMessage)m).destinations;

}

}

/\*\*

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

private class GenericMessage extends Message {

/\*\* Destinations. \*/

private List<Integer> destinations;

private Integer partner;

public GenericMessage(int source, List<Integer> destination, BitSet visitedNodes) {

super(source, -1, visitedNodes);

destinations = destination;

}

@Override

public Message copy() {

GenericMessage result = new GenericMessage(getSource(), destinations, getVisitedNodes());

result.setDestination(getDestination());

return result;

}

public int getPartner(final int number) {

return partner != null ? partner : Utils.getNextNode(number, getDestination(), d, Node.createRouteFilter(number, this, killed));

}

}

}

package org.mazur.toparch.router.one2all;

import java.util.ArrayList;

import java.util.BitSet;

import java.util.Collection;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import org.mazur.toparch.State;

import org.mazur.toparch.Utils;

import org.mazur.toparch.model.Message;

import org.mazur.toparch.model.Node;

import org.mazur.toparch.play.HopInfo;

import org.mazur.toparch.play.StepInfo;

import org.mazur.toparch.router.HopResolver;

import org.mazur.toparch.router.InputDataPanelFactory;

import org.mazur.toparch.router.LinkDescriptor;

import org.mazur.toparch.router.Router;

/\*\*

\* @version: $Id$

\* @author Roman Mazur (mailto: mazur.roman@gmail.com)

\*/

public class O2ARouter extends Router<O2AInputs> {

/\*\* Nodes. \*/

private List<GenericNode> nodes = null;

/\*\* Senders and busy nodes. \*/

private HashSet<GenericNode> senders, busyNodes = new HashSet<GenericNode>(),

markedNodes = new HashSet<GenericNode>();

private int step = 0;

@Override

protected InputDataPanelFactory<O2AInputs> createFactory() { return new O2AInputsFactory(); }

@Override

public String getName() { return "Один до всіх"; }

private void addSender(final GenericNode node, final List<LinkDescriptor> killed) {

node.refreshMessages(killed);

senders.add(node);

}

private GenericNode send(final Message m) {

GenericNode source = nodes.get(m.getSource());

source.sendCount++;

GenericNode dest = nodes.get(m.getDestination());

if (busyNodes.contains(dest)) { return null; }

busyNodes.add(dest);

dest.deniedPartners.add(source.getNumber());

return dest;

}

public String[][] formMDistrib() {

String[][] result = new String[nodes.size()][];

int index = 0;

for (Node node : nodes) {

result[index++] = node.getMessages();

}

return result;

}

@Override

public HashSet<GenericNode> getMarkedNodes() { return markedNodes; }

@Override

protected StepInfo next(final O2AInputs input) {

busyNodes.clear();

if (senders == null) {

senders = new HashSet<GenericNode>();

addSender(nodes.get(input.getSource()), input.getKilled());

}

markedNodes.addAll(senders);

StepInfo result = new StepInfo();

result.setStep(step++);

List<HopInfo> hops = new LinkedList<HopInfo>();

List<GenericNode> newSenders = new LinkedList<GenericNode>();

for (GenericNode s : senders) {

Message m = s.nextMessage();

if (m == null) { continue; }

GenericNode sender = send(m);

if (sender == null) { continue; }

sender.deniedPartners.add(input.getSource());

newSenders.add(sender);

HopInfo hop = new HopInfo();

hop.setSource(m.getSource());

hop.setDestination(m.getDestination());

hop.setDescription(getName());

hops.add(hop);

}

result.setHopsInfo(hops);

// clean senders

senders.addAll(newSenders);

List<GenericNode> toRemove = new LinkedList<GenericNode>();

for (GenericNode n : senders) {

if (!n.isSender()) {

toRemove.add(n);

} else {

n.refreshMessages(input.getKilled());

}

}

senders.removeAll(toRemove);

if (result.getHopsInfo().isEmpty()) { return null; }

return result;

}

@Override

public void reinit() {

markedNodes.clear();

int d = State.INSTANCE.getDimension();

int n = Utils.getNodesCount(d);

nodes = new ArrayList<GenericNode>(n);

for (int i = 0; i < n; i++) {

GenericNode node = new GenericNode();

node.setNumber(i);

nodes.add(node);

}

step = 0;

senders = null;

}

private static class GenericNode extends Node {

private static HopResolver[] resolvers = null;

static {

HopResolver[] r = STANDARD\_RESOLVERS;

resolvers = new HopResolver[] { r[3], r[0], r[2], r[1] };

}

private int sendCount = 0;

private Collection<Integer> deniedPartners = new HashSet<Integer>();

public Message nextMessage() { return getBuffers().isEmpty() ? null : getBuffers().first(); }

public boolean isSender() { return sendCount < resolvers.length; }

public void refreshMessages(final List<LinkDescriptor> killed) {

getBuffers().clear();

int dest = -1;

do {

while (isSender() && deniedPartners.contains(dest = resolvers[sendCount].getNext(getNumber()))) { sendCount++; }

if (isKilled(getNumber(), dest, killed)) { deniedPartners.add(dest); dest = -1; }

} while (isSender() && dest == -1);

if (dest >= 0 && isSender()) {

addMessage(new Message(getNumber(), dest, new BitSet(2)));

}

}

}

}

package org.mazur.toparch.router.one2one

import org.mazur.toparch.play.StepInfo;

import org.mazur.toparch.Utils.NearInfo;

import org.mazur.toparch.play.HopInfo;

import org.mazur.toparch.router.InputDataPanelFactory;

import java.util.HashSet;

import org.mazur.toparch.State;

import org.mazur.toparch.Utils;

import org.mazur.toparch.router.GroovyRouter;

import org.mazur.toparch.router.LinkDescriptor;

/\*\*

\* One2One.

\*/

class One2OneRouter extends GroovyRouter<One2OneInputs> {

/\*\* Current node. \*/

private int currentNode

/\*\* Step number. \*/

private int step

/\*\* Visited nodes. \*/

private HashSet<Integer> visited = new HashSet<Integer>()

@Override

public String getName() { return "Один до одного" }

@Override

public void reinit() {

println "Reinit"

currentNode = -1

step = 0

visited.clear()

}

private HopInfo hop(int s, int d, String descr) {

HopInfo hop = new HopInfo()

hop.source = s

hop.destination = d

hop.description = descr

return hop

}

private boolean checkLink(final int node, final List<LinkDescriptor> killed) {

for (LinkDescriptor ld in killed) {

if ((ld.getSource() == currentNode && ld.getDestination() == node)

|| (ld.getSource() == node && ld.getDestination() == currentNode)) {

return false

}

}

return true

}

private def routeInCluster(final int dst, final String msg, final int d, final One2OneInputs inputs) {

int cs = d << 1

int cnIndex = currentNode % cs

int distance = Utils.getInClusterDistance(cnIndex, dst, d)

int currentCluster = currentNode / cs

println "Route in cluster to $dst from $cnIndex, distance: $distance"

if (distance == 1 && !checkLink(currentCluster \* cs + dst, inputs.killed)) {

println "$dst is down"

return null

}

def partners = [(cnIndex + 1) % cs, (cnIndex + d) % cs, cnIndex ? cnIndex - 1 : cs - 1].sort() { a, b ->

Utils.getInClusterDistance(a, dst, d) <=> Utils.getInClusterDistance(b, dst, d)

}

println "Parteners: $partners"

int next = partners.find {

int node = currentCluster \* cs + it

return checkLink(node, inputs.killed) && !visited.contains(node)

}

if (next == null) { return null }

int prev = currentNode

currentNode = currentCluster \* cs + next

return [hop(prev, currentNode, msg + "; go to $currentNode")]

}

private def routeBetweenClusters(final int destCluster, final int d, final One2OneInputs input) {

int cs = d << 1

int currentCluster = currentNode / cs

int[] differentAxises = Utils.compareClusters(currentCluster, destCluster, d)

def transitionClusters = differentAxises.collect() {

int v = Utils.getDigit(destCluster, it)

return Utils.setDigit(currentCluster, it, v)

}

println "Possible transition clusters: $transitionClusters"

int currentNodeIndex = currentNode % cs

def closeJumpPoints = transitionClusters.collect { Utils.getNearInfo(currentCluster, it, d).getSource() }

closeJumpPoints = closeJumpPoints.sort() { a, b ->

Utils.getInClusterDistance(a, currentNodeIndex, d) <=> Utils.getInClusterDistance(b, currentNodeIndex, d)

}

closeJumpPoints += closeJumpPoints.collect { (it + (cs >> 1)) % cs } // other jump points

println "Jump points: $closeJumpPoints"

String msg = ""

def tryJumpPoints = { jumpPoints ->

for (int njp in jumpPoints) {

int jumpPoint = currentCluster \* cs + njp

int nextNode = Utils.getNearClusterConnection(jumpPoint, d)

println "current: $currentNode"

println "Try jump point node: $jumpPoint -> $nextNode"

if (visited.contains(nextNode)) {

msg += "$nextNode was visited."

println "$nextNode was visited."

continue

}

if (jumpPoint == currentNode) {

if (!checkLink(nextNode, input.killed)) {

msg += "Jump link to $nextNode is down."

println "Jump link to $nextNode is down."

continue

}

currentNode = nextNode

msg += "Jumping from $jumpPoint to $currentNode."

return [hop(jumpPoint, currentNode, msg)]

} else {

if (visited.contains(jumpPoint)) { continue }

msg += "Look for $jumpPoint to jump."

def res = routeInCluster(njp, msg, d, input)

if (res != null) { return res }

}

}

return null

}

def res = tryJumpPoints(closeJumpPoints)

if (res != null) { return res }

println "Cannot reach by near axises. Go far :)"

def goFarJP = []

cs.times() { if (!closeJumpPoints.contains(it)) { goFarJP += it } }

println "Try them: $goFarJP \n---------"

return tryJumpPoints(goFarJP)

}

@Override

protected StepInfo next(final One2OneInputs input) {

int destNode = input.destination

if (currentNode == destNode) { return null }

step++

int d = State.INSTANCE.dimension

int cs = d << 1

if (currentNode < 0) { currentNode = input.source }

visited.add currentNode

println "Routing $currentNode -> $destNode"

int currentCluster = currentNode / cs

int destCluster = destNode / cs

println "Clusters: $currentCluster -> $destCluster"

StepInfo result = new StepInfo()

result.step = step

if (currentCluster == destCluster) {

String msg = "Come closer to $destNode"

result.hopsInfo = routeInCluster(destNode % cs, msg, d, input)

} else {

result.hopsInfo = routeBetweenClusters(destCluster, d, input)

}

if (result.hopsInfo == null || result.hopsInfo.empty) {

println "Destination is unreachable"

return null

}

return result

}

@Override

protected InputDataPanelFactory<One2OneInputs> createFactory() { return new One2OneInputPanelFactory() }

}