ЗМІСТ

[ВСТУП 2](#_Toc470808013)

[РОЗДІЛ 1. СИнтез ТОПОЛОГІЧНОЇ ОРГАНІЗАЦІЇ 3](#_Toc470808014)

[1.1 Вибір оригінальної топологічної організації 3](#_Toc470808015)

[1.2 Математичний опис топології 5](#_Toc470808016)

[1.3 Програмний опис топології 6](#_Toc470808017)

[1.4 Визначення топологічних характеристик 7](#_Toc470808018)

[1.5 Порівняння результатів з характеристиками інших топологій 8](#_Toc470808019)

[1.6 Відображення топології на гіперкуб 14](#_Toc470808020)

[1.7 Висновки до розділу 1 15](#_Toc470808021)

[РОЗДІЛ 2. алгоритми маршрутизації 16](#_Toc470808022)

[2.1 Алгоритм маршрутизації one-to-one 16](#_Toc470808023)

[2.2 Алгоритм маршрутизації all-to-all 21](#_Toc470808024)

[2.3 Алгоритм маршрутизації one-to-all personalized 24](#_Toc470808025)

[2.4 Алгоритм маршрутизації one-to-all personalized 26](#_Toc470808026)

[2.5 Висновки до розділу 2 28](#_Toc470808027)

[РОЗДІЛ 3. завантаження задачі в розроблену топологію 29](#_Toc470808028)

[3.1 Опис алгоритму шифрування DES 29](#_Toc470808029)

[3.2 Реалізація шифрування 30](#_Toc470808030)

[3.3 Завантаження задачі в топологію 32](#_Toc470808031)

[3.4 Висновки до розділу 3 34](#_Toc470808032)

[ВИСНОВКИ 35](#_Toc470808033)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 37](#_Toc470808034)

[Додаток 1. Лістинг програмного забезпечення](#_Toc356265139) 37

ВСТУП

У сучасному світі комп’ютерні системи з багатьма процесорами будуються в основному на основі гіперкубічної чи мешової топології, оскільки вони визнані найбільш продуктивними та ефективними. Метою курсового проекту з дисципліни “Проектування комп’ютерних систем” є дослідження топології, яка не належить до класів, згаданим вище. Природно, в чистому вигляді жоден інший клас не зміг би зрівнятися, наприклад, з гіперкубом, проте їх гібридні комбінації можуть дозволити досягти високих показників ефективності. Для вибору топологічної організації дуже важливо розбиратися в їх перевагах та недоліках, що впливають на продуктивність системи в цілому.

У даній роботі розглянуто модифіковане бінарне дерево, кожен ярус якого з’єднаний двома кільцями. У першому розділі виконується аналіз даної топології за різними характеристиками (ступінь, діаметр, ціна і т. д.) та порівняння її з іншими топологічними організаціями. Враховуючи, що існує безліч алгоритмів, орієнтованих на класичні топології, виконано відображення в гіперкуб.

У другому розділі розглядається реалізація трьох видів маршрутизації: one-to-one, all-to-all та one-to-all personalized. Всі ці види маршрутизації є жорстко пов’язаними з топологією, тобто в них не використовуються алгоритми Дейкстри або Флойда. Подібна вузька спеціалізація дозволяє оптимізувати часові витрати на пересилання даних. Розроблений базовий набір програмного забезпечення, лістинг якого можна знайти в додатку.

У третьому розділі виконано завантаження реальної задачі в систему з даною топологією. В якості задачі обраний алгоритм шифрування DES, який використовує перетворення за допомогою мереж Фейстеля. Саме ці перетворення дуже зручні для завантаження на різні вершини в топології.

# СИнтез ТОПОЛОГІЧНОЇ ОРГАНІЗАЦІЇ

* 1. **Вибір оригінальної топологічної організації**

Основою досліджуваної топології є бінарне дерево. У зв’язку з тим, що характеристики дерева зазвичай значно гірше, ніж у топологій типу меш, решітка або гіперкуб, дана топологія модифікована додаванням кілець на кожному ярусі. На ярусі розташовані як звичайне кільце, так і кільце “через один”, що забезпечує хорошу швидкість пересилання в горизонтальному напрямку. Швидкість вертикального пересилання забезпечується структурою кластера, у якого верхня і нижня вершини з’єднані. Кластер зображений на рис. 1.1.



Рисунок 1.1 – Кластер системи

Нумерація вузлів в кластері виконується по колу за годинниковою стрілкою, починаючи з верхньої вершини. Надалі, номери вершин в кластері будуть називатися відносним індексом.

Масштабування топології виконується шляхом додавання нового ярусу. На рис. 1.2, 1.3, 1.4 зображені топології 1-го, 2-го і 3-го рівнів відповідно.



Рисунок 1.2 – Система на першому кроці масштабування



Рисунок 1.3 – Система на другому кроці масштабування



Рисунок 1.4 – Система на третьому кроці масштабування

Як видно з малюнків, на перших двох рівнях спостерігаються додаткові зв’язки в кластерах. Це кільцеві, які на даних рівнях з’єднують кластер, фактично, сам з собою.

* 1. **Математичний опис топології**

Створену топологію можна охарактеризувати математично так:

В таблиці 1.1 наведено матрицю суміжності для кластера.

Таблиця 1.1 – Матриця суміжності кластера

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |

Алгоритм створення матриці інцидентності для топології наступний:

1. Створюється матриця розміром N\*N;
2. Ініціалізувати список меж ярусу у вигляді FIFO черги.
3. Виконується цикл для кожного кластера:
   1. Помістити матрицю інцидентності кластера у вільний блок 6\*6 на діагоналі основної матриці.
   2. Зв’язати вершину з відносним індексом 0 з вершиною, отриманої зі списку меж ярусу.
   3. Помістити вершину з відносним індексом 3 до списку меж ярусу.
4. Для кожного ярусу виконати:
   1. Для кожного кластера ярусу виконати:
      1. Виконати кільцевий зв’язок вершини з відносним індексом 1 з вершиною наступного кластера з відносним індексом 5.
      2. Виконати кільцевий зв’язок вершини з відносним індексом 2 з вершиною кластера, що слідує за наступним з відносним індексом 4.
   2. **Програмний опис топології**

На мові програмування Java реалізація виглядатиме так:

**public** **byte**[][] topology(**final** **int** numClusters) {

**int** length = cluster.length;

**int** P = getP(numClusters);

**byte**[][] matrix = **new** **byte**[P][P];

**int** offsetX = 0, offsetY = 0;

**for** (**int** numCluster = 0; numCluster < numClusters; numCluster++) {

**for** (**int** i = offsetX, ii = 0; i < offsetX + length; i++, ii++) {

**for** (**int** j = offsetY, jj = 0; j < offsetY + length; j++, jj++) {

matrix[i][j] = cluster[ii][jj];

}

}

**for** (**int** i = 0; i < length; i++) {

nodes.add(**new** Node(numCluster \* length + i, numCluster,

detectLevel(numCluster)));

}

**int** edge = numCluster \* length + 3;

**if** (!edges.isEmpty()) {

**int** parent = edges.removeLast();

*setLink*(matrix, numCluster \* length, parent);

}

edges.addFirst(edge);

edges.addFirst(edge);

offsetX += length;

offsetY += length;

}

**for** (**int** i = 0; i < level(numClusters); i++) {

**for** (**int** j = 0; j < clustersInLevel(i); j++) {

**int** numCluster = numClusters(i - 1) + j;

**int** ring1 = numCluster \* length + 1;

**int** ring2 = (numClusters(i - 1) + (j + 1) % clustersInLevel(i))

\* length + 5;

*setLink*(matrix, ring1, ring2);

**int** doubleRing1 = numCluster \* length + 2;

**int** doubleRing2 = (numClusters(i - 1) + (j + 2) % clustersInLevel(i))

\* length + 4;

*setLink*(matrix, doubleRing1, doubleRing2);

}

}

**for** (**int** i = 0; i < matrix.length; i++) {

**for** (**int** j = 0; j < matrix[i].length; j++) {

**if** (matrix[i][j] != 0) {

nodes.get(i).addEdge(nodes.get(j));

}

}

}

**return** matrix;

}

* 1. **Визначення топологічних характеристик**

Серед основних топологічних характеристик виберемо степінь, діаметр, середній діаметр, топологічний трафік та ціну системи.

Степінь (S) – максимальне число дуг, інцедентних вершині в графі топології. Розраховується як максимальна сума одиниць в рядку або стовпці матриці суміжності.

Діаметр (D) – мінімальна відстань між двома найбільш віддаленими процесорами. Мінімальна відстань між вузлами знаходиться за допомогою алгоритму Флойда-Уоршелла. При цьому складається матриця відстаней між кожними вершинами, а потім шукається максимальне значення в ній.

Середній діаметр () визначається за формулою , де dij – мінімальна відстань між двома процесорами.

Топологічний трафік (T) визначається за формулою і характеризує завантаженість ліній зв’язку.

Ціна (C) визначає вартість створення системи та знаходиться за формулою .

Для визначення топологічних характеристик було написано програмне забезпечення, яке було запущено з різним параметром N, після чого було отримано інформацію про значення основних топологічних характеристик. Отримані результати були зведені до табл. 1.2.

Таблиця 1.2 – Основні характеристики системи з обраною топологією

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Кластерів | N | S | D |  | T | C |
| 1 | 6 | 3 | 2 | 1.43 | 0.96 | 36 |
| 3 | 18 | 5 | 5 | 2.74 | 1.09 | 450 |
| 7 | 42 | 5 | 7 | 3.81 | 1.52 | 1470 |
| 15 | 90 | 5 | 9 | 4.96 | 1.98 | 4050 |
| 31 | 186 | 5 | 12 | 6.46 | 2.59 | 11160 |
| 63 | 378 | 5 | 16 | 8.56 | 3.42 | 30240 |
| 127 | 762 | 5 | 20 | 11.26 | 4.5 | 76200 |
| 255 | 1530 | 5 | 24 | 14.43 | 5.77 | 183600 |
| 511 | 3066 | 5 | 28 | 17.92 | 7.17 | 429240 |
| 1023 | 6138 | 5 | 32 | 21.62 | 8.65 | 982080 |

* 1. **Порівняння результатів з характеристиками інших топологій**

Для об’єктивної оцінки характеристик даної топології, було виконано її порівняння з іншим, подібним деревом. Його кластер аналогічний (рис. 1.1), проте на 2 кроці масштабування система має вигляд, як на рис. 1.5.

Порівняльний аналіз виконується наступним чином:

1. Отримати дані про основні топологічні характеристики для різного розміру системи.
2. На основі табличних даних побудувати графіки залежності степеню, діаметру, середнього діаметру, ціни та трафіку від кількості вершин.
3. Загальний діапазон ділиться на кілька середніх за кількістю вимірів діапазонів.
4. Обчислити середні та нормалізовані значення кожної топологічної характеристики у всіх діапазонах.
5. Обчислити суму балів для кожного діапазону, на основі чого побудувати гістограму та зробити висновки про ефективність роботи системи на кожному діапазоні. Побудувати гістограму віддаленості від ідеалу (різниця набраних балів та мінімально можливої кількості балів).

Нормалізація проводиться наступним чином:

1. Степінь: серед значень топологій для одного діапазону було вибрано мінімальне значення – воно вважається оптимальним.
2. Мінімальна степінь – мінімальна ціна. Параметри топологій в даному діапазоні поділені на мінімальне значення.
3. Діаметр: аналогічно степені, знайдено мінімальне значення і на нього поділені всі значення для даного діапазону.
4. Середній діаметр: аналогічно степені, знайдено мінімальне значення і на нього поділені всі значення для даного діапазону.
5. Ціна: знайдено мінімальне значення і на нього поділені всі значення для даного діапазону.
6. Трафік: замість мінімального значення, знайдено значення найбільш наближене до оптимуму (тобто 1). На нього поділені всі інші значення.



Рисунок 1.5 – Дерево для порівняння на другому кроці масштабування

Топологічні характеристики цього дерева, зібрані за допомогою розробленого програмного забезпечення, представлені в табл. 1.3.

Таблиця 1.3 – Основні характеристики системи з топологією дерево

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | S | D |  | T | C |
| 6 | 3 | 3 | 1.6667 | 1.1111 | 54 |
| 13 | 4 | 5 | 2.4872 | 1.2436 | 260 |
| 22 | 4 | 7 | 3.2814 | 1.6407 | 616 |
| 33 | 4 | 9 | 4.0606 | 2.0303 | 1188 |
| 46 | 4 | 11 | 4.8300 | 2.4150 | 2024 |
| 61 | 4 | 13 | 5.5923 | 2.7962 | 3172 |
| 78 | 4 | 15 | 6.3497 | 3.1748 | 4680 |
| 97 | 4 | 17 | 7.1031 | 3.5515 | 6596 |
| 118 | 4 | 19 | 7.8535 | 3.9268 | 8968 |
| 141 | 4 | 21 | 8.6016 | 4.3008 | 11844 |
| 166 | 4 | 23 | 9.3478 | 4.6739 | 15272 |
| 193 | 4 | 25 | 10.092 | 5.0462 | 19300 |
| 222 | 4 | 27 | 10.836 | 5.4179 | 23976 |
| 253 | 4 | 29 | 11.578 | 5.7890 | 29348 |
| 286 | 4 | 31 | 12.319 | 6.1596 | 35464 |
| 321 | 4 | 33 | 13.060 | 6.5299 | 42372 |
| 358 | 4 | 35 | 13.800 | 6.8998 | 50120 |
| 397 | 4 | 37 | 14.539 | 7.2695 | 58756 |

За таблицями 1.2 і 1.3 були побудовані графіки для порівняння різних топологічних характеристик. Результати відображені на рис. 1.6-1.10.

Рисунок 1.6 – Порівняння степенів

Рисунок 1.7 – Порівняння діаметрів

Рисунок 1.8 – Порівняння середніх діаметрів

Рисунок 1.9 – Порівняння цін

Рисунок 1.10 – Порівняння трафіків

Таблиця 1.3 – Середні значення характеристик топології дерево

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Інтервали | S | D |  | C | T |
| 6..40 | 3.75 | 6 | 2.873975 | 529.5 | 0.4936 |
| 40..100 | 4 | 14 | 5.968775 | 4118 | 2.9844 |
| 100..400 | 4 | 28 | 11.20269 | 29542 | 5.6013 |

Таблиця 1.4 – Нормалізовані значення характеристик топології дерево

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Інтервали | S | D |  | C | T | ∑ | Віддаленість |
| 6..40 | 1 | 1.71429 | 1.3784 | 2.179 | 0.4815 | 6.75324 | 1,75324 |
| 40..100 | 1 | 1.75 | 1.3612 | 1.492 | 1.7054 | 7.30857 | 2,30857 |
| 100..400 | 1 | 2 | 1.4917 | 1.4271 | 1.864 | 7.78286 | 2,78286 |

Таблиця 1.5 – Середні значення характеристик оригінальної топології

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Інтервали | S | D |  | C | T |
| 6..40 | 4 | 3.5 | 2.085 | 243 | 1.025 |
| 40..100 | 5 | 8 | 4.385 | 2760 | 1.75 |
| 100..400 | 5 | 14 | 7.51 | 20700 | 3.005 |

Таблиця 1.6 – Нормалізовані значення характеристик оригінальної топології

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Інтервали | S | D |  | C | T | ∑ | Віддаленість |
| 6..40 | 1.06667 | 1 | 1 | 1 | 1 | 5.06667 | 0.06667 |
| 40..100 | 1.25 | 1 | 1 | 1 | 1 | 5.25 | 0.25 |
| 100..400 | 1.25 | 1 | 1 | 1 | 1 | 5.25 | 0.25 |

Рисунок 1.11 – Сумарна оцінка топологій по зонах

Рисунок 1.12 – Віддаленість топологій від ідеалу в діапазонах

* 1. **Відображення топології на гіперкуб**

Існують наступні методи для завантаження топологій в гіперкуб:

1. Звуження графу – кілька ребер вихідного графу відображаються на одне ребро стандартного. Недоліком методу є збільшення трафіку.
2. Розтягування графу – ребро вихідного графу відображається за допомогою кількох ребер стандартного. Недоліком є зростання часу.
3. Розширення графу. В цьому випадку одна вершина вихідного графу може відображатись за допомогою кількох вершин стандартного.

У даній роботі використовувалися другий і третій методи.

Топологія на 1 кроці масштабування зображена на рис.1.2, а другого – на рис. 1.3. Відображення цих топологій в гіперкуб показано на рис. 1.13-1.14.



Рисунок 1.13 – Відображення топології 1-го рівня в гіперкуб 3 порядку



Рисунок 1.14 – Відображення топології 2-го рівня в гіперкуб

* 1. **Висновки до розділу 1**

В цьому розділі була представлена оригінальна топологія, заснована на кластері, зображеному на рис. 1.1. Завдяки характерним міжкластерним зв’язкам, топологія дуже схожа на класичне дерево.

Було представлено можливість необмеженого масштабування топології. Збільшення кількості зв’язків на пізніших кроках масштабування забезпечує стійкість до відмов, проте збільшує ціну топології.

З рис. 1.2-1.4 очевидно недолік кластерів – верхній ярус поєднаний з нижнім допомогою лише одного дуального зв’язку для кожного кластера. Таким чином, вихід з ладу вершини з відносним індексом 3 суттєво вплине на продуктивність системи, а вихід з ладу безлічі таких вершин може зупинити роботу системи взагалі.

За допомогою розробленого програмного забезпечення було визначено характеристики оригінальної топології та дерева для різної кількості вузлів.

Як видно з табл. 1.2, степінь є фактично постійною величиною. Діаметр топології звичайний для організацій подібного роду. Середній діаметр менше середнього для класичних топологій. Ціна зростає експоненціально, оскільки для її розрахунків діаметр множиться на кількість всіх ребер та на кількість всіх вершин.

В сумі очевидно, що запропонована топологія виграє у дерева по всіх, окрім степеню, параметрах та значно менше віддалена від ідеалу.

В розділі також було проілюстровано відображення запропонованої топології в гіперкуб.

# алгоритми маршрутизації

* 1. **Алгоритм маршрутизації one-to-one**

Алгоритм one-to-one виконує пересилання даних з вихідної вершини в зазначену за мінімальний час. В якості оцінки часу береться кількість переходів між вершинами, тобто мається на увазі, що час виконання одного переходу становить один такт. Досліджувана топологія представляє собою дерево, а тому маршрутизація легко розбивається на процедури.

**Формалізований алгоритм:**

1. Для кожної вершини визначити наступні параметри: індекс, відносний індекс, номер ярусу та індекс кластера.
2. Помістити покажчик на початкову вершину.
3. Виконати цикл:
   1. Якщо індекс покажчика збігається з індексом одержувача, то вийти з циклу.
   2. Якщо індекс кластера покажчика збігається з індексом кластера отримувача, то виконати процедуру передачі в кластері і помістити покажчик на її результат.
   3. Якщо ярус покажчика збігається з ярусом одержувача, то виконати процедуру передачі даних по ярусу і помістити покажчик на її результат.
   4. Якщо ярус покажчика більше ярусу одержувача, то:
      1. Виконати процедуру переходу по кластеру для переміщення до вершини в кластері з відносним індексом 0.
      2. Виконати перехід у зовнішній кластер, використовуючи вершину з відносним індексом 3.
      3. Перемістити покажчик у вершину, отриману раніше.
   5. Якщо ярус покажчика менше ярусу одержувача, то:
      1. Виконати процедуру пошуку батьківського вузла для одержувача.
      2. Виконати процедуру переміщення по ярусу з вузла, на який вказує покажчик, до батьківського.
      3. Виконати процедуру переміщення по кластеру з вузла, на який вказує покажчик, у вершину того ж кластера з відносним індексом 3.
      4. Виконати процедуру пошуку батьківських вузлів для одержувача, використовуючи збільшений на 1 номер ярусу.
      5. Виконати перехід з вузла, в якому покажчик, до батьківського.
      6. Присвоїти вказівнику результат, отриманий на минулому етапі.

**Алгоритм процедури пошуку батьківського вузла:**

1. Якщо ярус необхідної вершини збігається з параметричним ярусом, то повернути вершину з того ж кластера з відносним індексом 0.
2. Виконати цикл:
   1. Знайти внутрішній перехід в кластер з відносним індексом 0.
   2. Знайти зовнішній перехід в кластер з відносним індексом 3.
   3. Якщо ярус отриманої вершини збігається з параметричним ярусом, то вийти з циклу.

**Алгоритм процедури переміщення по ярусу:**

1. Поки кластер покажчика не збігається з кластером одержувача, виконати:
   1. Розрахувати відстань до цільового кластера при русі вправо і вліво.
   2. Якщо відстань вправо менше або дорівнює відстані вліво, то:
      1. Якщо відстань рівна 1:
         1. Виконати процедуру переміщення по кластеру в вершину з відносним індексом 1.
         2. Виконати переміщення по зовнішньому зв’язку до вершини з відносним індексом 5.
         3. Присвоїти вказівнику результат, отриманий раніше.
      2. Якщо відстань більше 1:
         1. Виконати процедуру переміщення по кластеру в вершину з відносним індексом 2.
         2. Виконати переміщення по зовнішньому зв’язку у вершину з відносним індексом 4.
         3. Присвоїти вказівнику результат, отриманий раніше.
   3. Якщо відстань вліво менше відстані вправо:
      1. Якщо відстань рівна 1:
         1. Виконати процедуру переміщення по кластеру в вершину з відносним індексом 5.
         2. Виконати переміщення по зовнішньому зв’язку в вершину з відносним індексом 1.
         3. Присвоїти вказівнику результат, отриманий раніше.
      2. Якщо відстань більше 1:
         1. Виконати процедуру переміщення по кластеру в вершину з відносним індексом 4.
         2. Виконати переміщення по зовнішньому зв’язку в вершину з відносним індексом 2.
         3. Присвоїти вказівнику результат, отриманий раніше.

**Алгоритм процедури переміщення по кластеру:**

1. Запам’ятати початкову вершину в покажчик.
2. Виконати цикл:
   1. Якщо індекс покажчика дорівнює індексу цільової вершини, то вийти з циклу.
   2. Якщо список сусідів покажчика містить цільову вершину, то виконати перехід і присвоїти покажчику цільову вершину.
   3. Залежно від відносного індексу покажчика виконати дії:
      1. Якщо індекс дорівнює 2 й цільова вершина має відносний індекс 5 і має зв’язок 4-5, то перейти в вершину з відносним індексом 4.
      2. Якщо індекс дорівнює 4 й цільова вершина має відносний індекс 1 і має зв’язок 1-2, то перейти в вершину з відносним індексом 2.
      3. Якщо індекс дорівнює 2, 4 або 0, то виконати перехід в вершину з відносним індексом 3.
      4. Якщо індекс дорівнює 1 й цільова вершина має відносний індекс 4 і має зв’язок 4-2, то перейти в вершину з відносним індексом 2.
      5. Якщо індекс дорівнює 5 й цільова вершина має відносний індекс 2 і має зв’язок 2-4, то перейти в вершину з відносним індексом 4.
      6. Якщо індекс дорівнює 1, 5 або 3, то виконати перехід в вершину з відносним індексом 0.

Наведемо приклад роботи алгоритму. Нехай необхідно передати повідомлення з вузла 20 у вузол 13 у системі з кроком масштабування 3.

Sending from node #20 to node #13

20 -> 21

21 -> 18

18 -> 9

9 -> 6

6 -> 7

7 -> 17

17 -> 12

12 -> 13

Data arrived to destination

Результат роботи відображено на рис. 2.1.



Рисунок 2.1 – Виконання алгоритму one-to-one у системі без відмов

Реалізація стійкості до відмов зроблена за допомогою модифікації кожної процедури на випадок відмови. Якщо неможливо вибрати один із шляхів, то програма знаходить обхідний шлях між двома точками.

Приклад роботи з вимкненою вершиною 6:

Sending from node #20 to node #13

20 -> 21

21 -> 18

18 -> 9

9 -> 8

8 -> 7

7 -> 17

17 -> 12

12 -> 13

Data arrived to destination

Результат роботи алгоритму зображено на рис. 2.2.



Рисунок 2.2 – Виконання алгоритму one-to-one у системі з 1 відмовою

* 1. **Алгоритм маршрутизації all-to-all**

Алгоритм маршрутизації all-to-all призначений для пересилки даних всіх вершин всім іншим в топології. Враховуючи особливість деревовидної топології, раціональніше всього відправляти дані в ключові вершини, які є сполучними для різних кластерів. При розсилці слід врахувати, що перед відсиланням даних треба запитати у кінцевого вузла, чи вони вже є в нього, і, якщо є, то вибрати іншу вершину з можливих. Ідеальним варіантом поширення інформації була б прогресія (на 1 кроці дані отримує 1 вузол, на 2 – 2, на 3 кроці – 4 і т. д.), але це можливо тільки при виконанні відправлення даних з строго певних вершин у графі. В реальності, прогресія спадає на 4-5 кроці, що пов’язано зі ступенем топології.

При розробці даного алгоритму маршрутизації використовувався алгоритм one-to-all.

Формалізований алгоритм маршрутизації:

1. Для кожної вершини виконати:
   1. Визначити відносний індекс вершини, з якої відправляються дані.
   2. Поки всі вершини не отримали дані, виконати:
      1. Скласти список вершин, готових до відправлення.
      2. Прибрати зі списку ті вершини, у яких всі сусіди отримали дані.
      3. Для кожної вершини зі списку виконати пересилання, залежно від її індексу:
         1. Якщо відносний індекс дорівнює 0, то переслати дані вище стоячому кластеру в вершину з відносним індексом 3. Для цього визначаються сусіди з подібним індексом, що не належать поточному кластеру. Якщо цільова вершина вже отримала дані, то вибрати вершину з мінімальним індексом із сусідніх вершин.
         2. Якщо відносний індекс дорівнює 1 або 5, то переслати дані в вершину з відносним індексом 0 в даному кластері. Якщо цільова вершина вже отримала дані, то вибрати вершину з мінімальним індексом із сусідніх вершин.
         3. Якщо відносний індекс дорівнює 3, то переслати дані нижче стоячому кластеру в вершину з відносним індексом 0. Для цього визначаються сусіди з подібним індексом, що не належать поточному кластеру. Якщо доступно 2 таких кластери, то вибирається вершина з більшим індексом. Якщо цільова вершина вже отримала дані, то вибрати вершину з максимальним індексом із сусідніх вершин.
         4. Якщо відносний індекс дорівнює 2 або 4, то переслати дані в вершину з відносним індексом 3 в даному кластері. Якщо цільова вершина вже отримала дані, то вибрати вершину з максимальним індексом із сусідніх вершин.
      4. Якщо список виконаних пересилань порожній, то в топології є недосяжні вершини. Тому треба передати інформацію про помилку з номерами вершин, в які неможливо передати дані.

Наведемо приклад роботи алгоритму для топології 2-го рівня масштабування. В якості кроку будемо рахувати всі пересилання, що виконуються паралельно.

Оскільки відображення маршрутизації all-to-all дуже громіздке, для економії наведемо лише його частину, наприклад розсилання даних з вузла 16 всім іншим. Інші передачі відбуваються аналогічним чином.

Sending from node #16

Step 1

1: 16-->15

Step 2

2: 16-->14

Step 3

3: 14-->13

4: 16-->17

Step 4

5: 13-->12

6: 17-->7

Step 5

7: 7-->6

8: 12-->3

9: 13-->11

Step 6

10: 3-->4

11: 6-->9

12: 7-->8

13: 11-->10

Step 7

14: 3-->2

15: 4-->5

Step 8

16: 2-->1

17: 3-->0

Step 9

All nodes received data



Рисунок 2.3 – Виконання алгоритму one-to-all у системі без відмов

Нехай відмовили вершини 3, 7, 14. Тоді повідомлення їм не відправляється, а шлях набуде вигляду, зображеного на рис. 2.4.

Оскільки відображення маршрутизації all-to-all дуже громіздке, для економії наведемо лише його частину, наприклад розсилання даних з вузла 16 всім іншим. Інші передачі відбуваються аналогічним чином.

Очевидно, що через відмову вузла 3, весь перший кластер не зміг отримати дані. Тому для систем з такою топологією працездатність цих вузлів грає ключову роль.

Sending from node #16

Step 1

1: 16-->15

Step 2

2: 16-->17

Step 3

3: 17-->12

Step 4

4: 12-->13

Step 5

5: 13-->11

Step 6

6: 11-->6

Step 7

7: 6-->9

8: 11-->10

Step 8

9: 9-->8

Step 9

Nodes failed to recieve data:

0 1 2 3 4 5



Рисунок 2.4 – Виконання алгоритму one-to-all у системі з 3 відмовами

* 1. **Алгоритм маршрутизації one-to-all personalized**

Алгоритм маршрутизації one-to-all personalized призначений для пересилання персоналізованих даних з однієї вершини всім іншим в топології. Фактично він базується на алгоритмі one-to-one, але слід враховувати, що неможливо одночасно передавати кілька пакетів даних по одному зв’язку. Тому кожна вершина повинна містити список пакетів і на кожному кроці відсилати по одному на кожний необхідний зв’язок. Цей список є чергою, тобто перший пакет, що прийшов, повинен вилучатись з нього також першим.

Формалізований алгоритм маршрутизації наступний:

1. Додати вершину відправник у список активних вершин.
2. Додати повідомлення для всіх вершин, крім відправника, у повідомлення.
3. В список списків пасивних сусідів додати сусідів вершини відправника.
4. Поки є вершини, які не отримали дані, виконати:
   1. Якщо немає пасивних сусідів поточної активної вершини, тоді:
      1. Видалити з активних вершин поточну активну вершину.
      2. Зі списку пасивних сусідів видалити список, що належить поточній активній вершині.
   2. Інакше:
      1. Вибрати першу пасивну вершину.
      2. В оброблені вершини на поточному рівні додати поточну активну вершину.
      3. З повідомлення видалити відправника і всіх сусідів, крім обраного.
      4. Передати повідомлення пасивній вершині.
      5. У список активних вершин наступного рівня додати пасивну вершину.
      6. У список списків пасивних сусідів додати сусідів пасивної вершини.
      7. Якщо немає пасивних сусідів поточної активної вершини, тоді:
         1. Видалити з активних вершин поточну активну вершину.
         2. Зі списку пасивних сусідів видалити список, що належить поточній активній вершині.
   3. Якщо всі вершини зі списку активних вершин містяться в списку оброблених вершин, тоді:
      1. Збільшити рівень на 1.
      2. У список активних вершин додати всі вершини зі списку активних вершин наступного рівня.
      3. Очистити список активних вершин наступного рівня.

Наведемо приклад роботи алгоритму для системи на 2 рівні масштабування. На рис. 2.5 відображено розсилання повідомлень, що відбувається з вершини 6.

Sending from node #6

Step 1

1: 6-->3 (0, 1, 2, 3, 4, 5, 12, 13, 14)

Step 2

2: 3-->12 (0, 1, 2, 4, 5, 12, 13, 14, 15)

3: 6-->7 (7, 8, 9, 10, 11, 16, 17)

Step 3

4: 3-->4 (0, 1, 2, 4, 5)

5: 6-->9 (9, 10)

6: 7-->17 (16, 17)

7: 12-->13 (13, 14)

Step 4

8: 3-->2 (0, 1, 2)

9: 4-->5 (5)

10: 6-->11 (11)

11: 7-->8 (8)

12: 9-->10 (10)

13: 12-->15 (15)

14: 13-->14 (14)

15: 17-->16 (16)

Step 5

16: 2-->1 (1)

17: 3-->0 (0)

Step 6

All nodes received data



Рисунок 2.5 – Виконання алгоритму one-to-all personalized у системі без відмов

* 1. **Алгоритм маршрутизації all-to-one single node accumulation**

В алгоритмі all-to-one single node accumulation всі вузли одночасно відправляють повідомлення, одержувачем якого є один і той же вузол. Завдання, яке вирішує даний алгоритм - збір даних від усіх вузлів в вузлі призначення. Для вирішення даної задачі можуть використовуватись декілька підходів. Перший з них це використання all-to-all але даний підхід є дуже неоптимальним і врази програє наступному. Другий підхід це використання паралельної роботи для всіх вузлів алгоритму one-to-one даний підхід блокує певні вузли на довгий час але зменшує кількість передач даних.

Для оптимального вирішення даного завдання доцільно використовувати алгоритм one-to-one для кожного вузла але з акумулюванням пакетів від вузлів які повинні пройти через даний вузол. Припустимо що підчас запуску all-to-one всі вузли одночасно починають передавати дані у вузол призначення. Так як вузол одночасно може приймати і передавати лише по одному каналу то зробимо припущення що коли вузол вже приймає повідомлення всі іншим вузлам які намагаються передати йому пакет буде відповідати що він занятий і не може прийняти повідомлення. Також припустимо що кожен вузол періодично опитує своїх сусідів і знає які з них вийшли з ладу. Кожен вузол намагається передати пакет вузлу сусіду з найменшим значенням ідентифікатора.

Формалізований алгоритм маршрутизації наступний:

1) Всі вузли одночасно ініціюють пердачу пакетів вузлу отримувачу

2) Кожен з вузлів вибирає для адресата пакета оптимальний перехід в один з сусідніх вузлів за допомогою алгоритма one to one.

3) Якщо вузол вже приймає пакет(позначимо його Х) то він відповідає всім вузлам, які намагються передати йому дані, що він в даний момент не може прийняти пакет.

3.1) Вузли які отримали повідомлення про зайнятість вузла Х акамулюють всі інші повідомлення які їм надходять.

3.2) При звільнені вузла Х він починає приймати дані від сусіда з найменшим значенням ідентифікатора

4) Вузол який немає пакетів для передачі і черга на прийняття даних в нього порожня завершує свою частину роботи в даному алгоритмі.

Приклад роботи алгоритму для P = 2, цільова вершина – 3 також, несправні вершини: 4, 10

Приклад роботи програми:

Step 1:

5 -M5-> 0

0 -M3 -> 3

1 -M1-> 2

11 -M11-> 6

8 -M8-> 7

15 -M15-> 12

16 -M16->17

14 -M14->13

Step 2:

0 -M5-> 3

7 -M7,M8-> 6

12 -M13,M14-> 12

Step 3:

2 -M1,M2-> 3

9 -M9-> 6

17 -M13,M17-> 12

Step 4:

6 -> M7,M8,M9 -> 3

Step 5:

12 -M13,M14,M15,M16,M17-> 3

* 1. **Висновки до розділу 2**

В даному розділі розроблено три стійкі до відмов алгоритми маршрутизації для запропонованої топології: one-to-one, all-to-all, one-to-all personalized.

Маршрутизація даної топології тісно пов’язана з синтезом та складається з двох частин: між та всередині кластера. В якості оцінки часу береться кількість переходів між вершинами.

Алгоритм one-to-one виконує пересилання даних з початкової вершини в зазначену за мінімальний час.

Алгоритм маршрутизації all-to-all призначений для пересилки даних всіх вершин всім іншим в топології. При розробці даного алгоритму маршрутизації використовувався алгоритм one-to-all.

Алгоритм one-to-all personalized призначений для передачі персоналізованих даних з однієї вершини всім іншим в топології. Фактично він базується на алгоритмі one-to-one, але кожній вершині відправляються пакети з різним вмістом.

Недоліком представлених алгоритмів є те, що вони не знаходять найкоротші шляхи. В той же час, їх перевагою є значна стійкість до відмов.

# завантаження задачі в розроблену топологію

Для завантаження в систему з топологією типу “дерево” добре підходять ті задачі, що мають яскраво виражені паралельні ділянки з залежностями по даних від результатів попередніх розрахунків. Це дозволяє виконувати спуск по дереву і розділяти поточний етап виконання завдання на дрібніші підзадачі. Подібний підхід часто використовується в різних алгоритмах шифрування, зокрема в алгоритмі DES.

* 1. **Опис алгоритму шифрування DES**

DES (англ. *Data Encryption Standard*) – це симетричний алгоритм шифрування даних, розроблений в 1970-80х, що з часом набув міжнародного застосування. Даний алгоритм є блоковим шифром. Щоб зрозуміти, як працює DES, необхідно розглянути принцип роботи блочного шифру, на прикладі мережі Фейстеля.

Вхідними даними для блочного шифру служать блок розміром *n* біт і *k*-бітний ключ. На виході, після застосування шифрувального перетворення, виходить *n*-бітний зашифрований блок, причому незначні відмінності вхідних даних, як правило, призводять до істотної зміни результату. Блокові шифри реалізуються шляхом багаторазового застосування до блоків вхідного тексту деяких базових перетворень.

Серед цих базових перетворень виділяють:

1. Складне перетворення на одній локальній частині блоку.
2. Просте перетворення між частинами блоку.

Оскільки перетворення виконується по блоках, окремим кроком потрібно виконати поділ вхідних даних на блоки необхідного розміру. При цьому, незалежно від формату вихідних даних: чи то текстові документи, зображення або інші файли, вони повинні бути перетворені в бінарний вигляд і тільки після цього розбиті на блоки. Все вищезазначене може здійснюватися як програмними, так і апаратними засобами.

Перетворення мережею Фейстеля – це перетворення над векторами (блоками), що представляють собою ліву і праву половини регістра зсуву. В алгоритмі DES використовуються пряме перетворення мережею Фейстеля при виконанні шифрування (рис. 3.1) та зворотне – при дешифруванні (рис. 3.2).

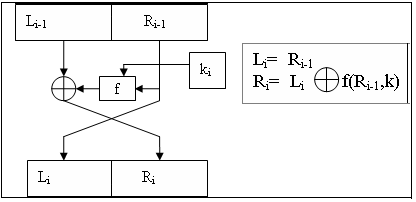


Рисунок 3.1 – Пряме перетворення Фейстеля

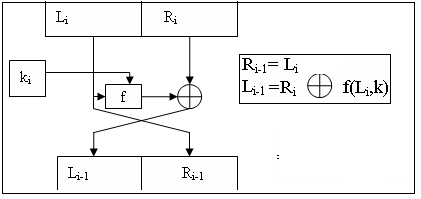


Рисунок 3.2 – Зворотне перетворення Фейстеля

* 1. **Реалізація шифрування**

Алгоритм роботи DES не дуже складний та складається з кількох етапів:

1. Початкова перестановка.
2. Цикли шифрування.
3. Мережа Фейстеля.
4. Генерація ключів.
5. Кінцева перестановка.

Детальний алгоритм роботи DES з усіма необхідними перетвореннями наведено на рис. 3.3.

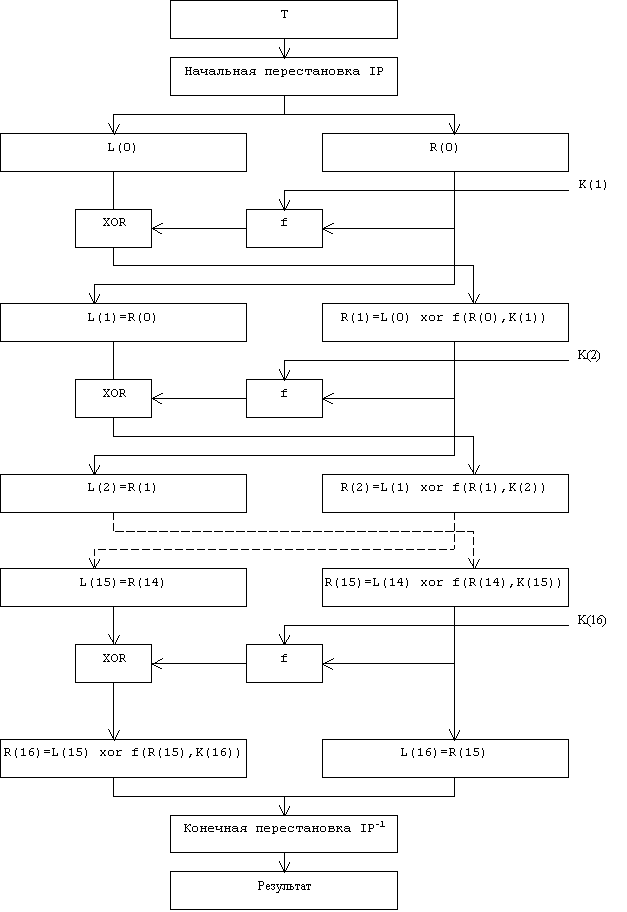
****

Рисунок 3.3 – Алгоритм DES

Код шифрування на мові Java.

private static BigInteger function(BigInteger right, BigInteger key) {

int ind = 0;

BigInteger ER = BigInteger.ZERO;

for (int i : ER) {

if (key.testBit(i - 1)) {

ER = ER.setBit(ind);

} else {

ER = ER.clearBit(ind);

}

ind++;

}

ER = ER.xor(key);

BigInteger[] B = new BigInteger[8];

BigInteger[] BB = new BigInteger[8];

for (int i = 7; i >= 0; i--) {

B[i] = BigInteger.ZERO;

for (int j = 0; j < 6; j++) {

if (ER.testBit(j)) {

B[i] = B[i].setBit(j);

} else {

B[i] = B[i].clearBit(j);

}

}

ER.shiftRight(6);

int a = 0;

if (B[i].testBit(0)) {

a++;

}

if (B[i].testBit(5)) {

a += 2;

}

int b = 0;

if (B[i].testBit(1)) {

b++;

}

if (B[i].testBit(2)) {

b += 2;

}

if (B[i].testBit(3)) {

b += 3;

}

if (B[i].testBit(4)) {

b += 4;

}

BB[i] = new BigInteger(String.valueOf(S[i \* 17 \* 4 + a \* 17 + b + 1]));

}

BigInteger f = BigInteger.ZERO;

for (int i = 0; i < 8; i++) {

for (int j = 3; j >= 0; j--) {

if (BB[i].testBit(j)) {

f.setBit(j);

} else {

f.clearBit(j);

}

}

f = f.shiftLeft(4);

}

ind = 0;

BigInteger result = BigInteger.ZERO;

for (int i : P) {

if (key.testBit(i - 1)) {

result = result.setBit(ind);

} else {

result = result.clearBit(ind);

}

ind++;

}

return result;

}

* 1. **Завантаження задачі в топологію**

Етапи 2-4 в алгоритмі DES можуть бути легко розпаралелені, що й було виконано для даної топології. Завантаження задачі полягає в наступному:

1. Завантажити у кореневій кластер вхідні дані та виконати початкову перестановку.
2. Виконати передачу даних алгоритмом one-to-all personalized.
3. Виконати другий етап шифрування на 16 вузлах системи.
4. Передати дані для обчислення в мережах Фейстеля. Оскільки всередині мережі виконується багато операцій, то маючи достатню кількість вузлів, їх також можна виконувати паралельно.
5. Зібрати результат обчислення за допомогою алгоритму one-to-one.
6. Виконати кінцеву перестановку.

Як вже було вказано, розсилання даних виконується за допомогою алгоритму one-to-all personalized, а збір результату – one-to-one. Таким чином, розпаралелювання дозволяє використовувати DES в декількох режимах: для шифрування, дешифрування або формування хешу.

Необхідно навести приклад виконання задачі, тобто промоделювати роботу алгоритму на системі з обраною топологією. Приймемо, що система має крок масштабування 2, тобто 16 вузлів, вхідні дані вводяться у 3, вага пересилання дорівнює 1, а вага операції – 10.

Тоді на початкове розсилання даних за допомогою алгоритму one-to-all personalized буде витрачено 5 тактів, на початкову та кінцеву перестановку ще по 10, а на паралельні обчислення 10. Багато часу витратиться на послідовний збір результату алгоритмом one-to-one – аж 33 (5 пересилань довжиною такт, 8 – на передачі з часом 2 та 4 – на шлях, довжиною три такти).

Отже, отримано результати, що сумарний час виконання Tp = 10 + 5 + 10 + 33 + 10 = 68.

Також необхідно підрахувати час виконання послідовного варіанту програми: Ts = 10 + 10\*16 + 10 = 180.

Виходячи з цього, можна розрахувати коефіцієнти прискорення та ефективності роботи системи.

Отже, коефіцієнт прискорення дорівнює , а коефіцієнт ефективності рівний .

* 1. **Висновки до розділу 3**

Для курсового проекту була вибрана задача виконання шифрування DES, що є дуже важливою для всіх областей, пов’язаних з криптографією.

Перш за все, було наведено необхідні теоретичні відомості та алгоритми.

Для реалізації завантаження задачі в систему використовувались алгоритми маршрутизації. Так, початкове розсилання даних виконується за допомогою алгоритму one-to-all personalized, а збір результату – one-to-one. Це дозволяє на практиці перевірити їх працездатність для відправлення вхідних даних та отримання проміжних та фінальних результатів обчислень.

При завантаженні задачі в систему з оригінальною топологією, було зроблено оцінку часу виконання алгоритму для конкретної розмірності, а також підраховані коефіцієнти прискорення та ефективності.

Отримані значення коефіцієнта прискорення та коефіцієнта ефективності далекі від ідеальних. Це можна пояснити значною кількістю передач даних. При збільшенні розмірності задачі та достатній пропускній здатності каналів, ця частина програми може бути значно зменшена.

ВИСНОВКИ

В ході виконання курсового проекту було розглянуто топологію на основі бінарного дерева, кожен ярус якого є також подвійним кільцем. Також показано можливість необмеженого масштабування системи.

За результатами порівняльного аналізу даної топологічної організації з деревом, було виявлено, що вона є оптимальнішою. В сумі очевидно, що запропонована топологія виграє у дерева по всіх, окрім степеню, параметрах та значно менше віддалена від ідеалу. Як видно з таблиць результатів, степінь є фактично постійною величиною. Діаметр топології звичайний для організацій подібного роду. Середній діаметр менше середнього для класичних топологій, а ціна зростає експоненціально.

Для забезпечення сумісності з алгоритмами, розробленими для класичних систем, виконано відображення даної топології в гіперкуб. Топологія на 1 кроці масштабування розміщується в гіперкубі 3-го порядку, а на другому – в гіперкубі 5-го порядку. Подальший ріст схожий з завантаженням звичайного бінарного дерева, тобто порядок зростає пропорційно збільшенню рівня.

У роботі розглянуто три види маршрутизації для даної топології: one-to-one, all-to-all, one-to-all personalized. Алгоритми є вузькоспеціалізованими і спрямованими на швидку роботу саме в даній топології. Маршрутизація даної топологічної організації тісно пов’язана з синтезом та складається з двох частин: між та всередині кластера. В якості оцінки часу береться кількість переходів між вершинами.

Недоліком представлених алгоритмів є те, що вони не знаходять найкоротші шляхи. В той же час, їх перевагою є значна стійкість до відмов.

В останньому розділі виконано завантаження задачі в систему з обраною топологією. В якості завдання було обрано алгоритм шифрування DES. При реалізації завантаження використовувалися алгоритми маршрутизації one-to-all personalized для розсилки вхідних даних і one-to-one для збору проміжних і фінального результатів.

При завантаженні задачі в систему з оригінальною топологією, було зроблено оцінку часу виконання алгоритму для конкретної розмірності, а також підраховані коефіцієнти прискорення та ефективності.

Також в ході роботи було розроблено програмне забезпечення для перевірки і демонстрації роботи створених алгоритмів маршрутизації та дослідження топологічних параметрів комп’ютерних систем.

Програмне забезпечення розроблялося з використанням платформи Java. До недоліків її використання можна віднести необхідність встановлення додаткового програмного забезпечення на комп’ютер користувача.

Проте перевагою даної методики є те, що, по-перше, платформа дозволяє запускати розроблений продукт на будь-якій операційні системі, що містить встановлений пакет JRE і, по-друге, дозволяє значно скоротити час розробки завдяки використання готових компонентів. Так, для відображення графа обчислювальної системи було використано бібліотеку JGraphX.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. JGraphX [Електронний ресурс] – Режим доступу до публ. : https://github.com/jgraph/jgraphx
2. Белов В. В., Воробьев Е. М., Шаталов В. Е. Теория графов: Учебн. пособие для вузов. – М. : Радио и связь, 1984. – 248 с.
3. Конспект лекцій з курсу “Проектування комп’ютерних систем”. Лектор – Луцький Г. М.
4. Симметричные криптосистемы [Електронний ресурс] – Режим доступу до публ. : http://ru.wikipedia.org/wiki/Симметричные криптосистемы
5. Хорстманн К., Корнелл Г. Java 2. Библиотека профессионала, том 1. Основы. = Core Java 2, Volume I – Fundamentals. – 8-е изд. – М. : Вильямс, 2008. – 816 с.
6. Шифрование DES [Електронний ресурс] – Режим доступу до публ. : http://habrahabr.ru/post/99376/
7. Эккель Б. Философия Java = Thinking in Java. – 3-е изд. – СПб. : Питер, 2003. – 976 с.