Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет прикладної математики
Кафедра системного програмування та спеціалізованих комп’ютерних систем

Комплексна лабораторна робота

з дисципліни:
“ Тестування, надійність, контроль та діагностика комп’ютерних систем ”
на тему:
« Оцінка надійності багатомодульних відмовостійких систем»

Виконали:
студенти групи КВ-41мн
Пилипченко Б.О. та
Безруков І. О.
Перевірив:
проф. Романкевич В. О.

Київ-2024

Зміст

1. Завдання (варіант 17) 3
2. Опис програмного рішення 5
3. Дослідження надійності 12
3.1 Початкова схема 15
3.2 Модифікація 1: 7-7-7-8-8 17
3.3 Модифікація 2: з’єднання елементів 19
3.4 Модифікація 3: 24 елементи (d9) 20
3.5 Модифікація 4: 25 елементів (d10) 22
3.6 Модифікація 5: 27 елементів (c7, c8) 24
3.7 Модифікація 6: 29 елементів (a3, a4) 26
4. Додаткове тестування швидкодії 29
5. Ручні обчислення деяких BCC для початкової схеми та модифікації 1 33
Метод виконання ручних розрахунків 33
Як виводилися правила 33
Перерозподіл навантаження 38
Початкова схема 38
Модифікація 1 39
Висновки 42

1. Завдання (варіант 17)

Для виконання лабораторної роботи було обрано варіант 17:

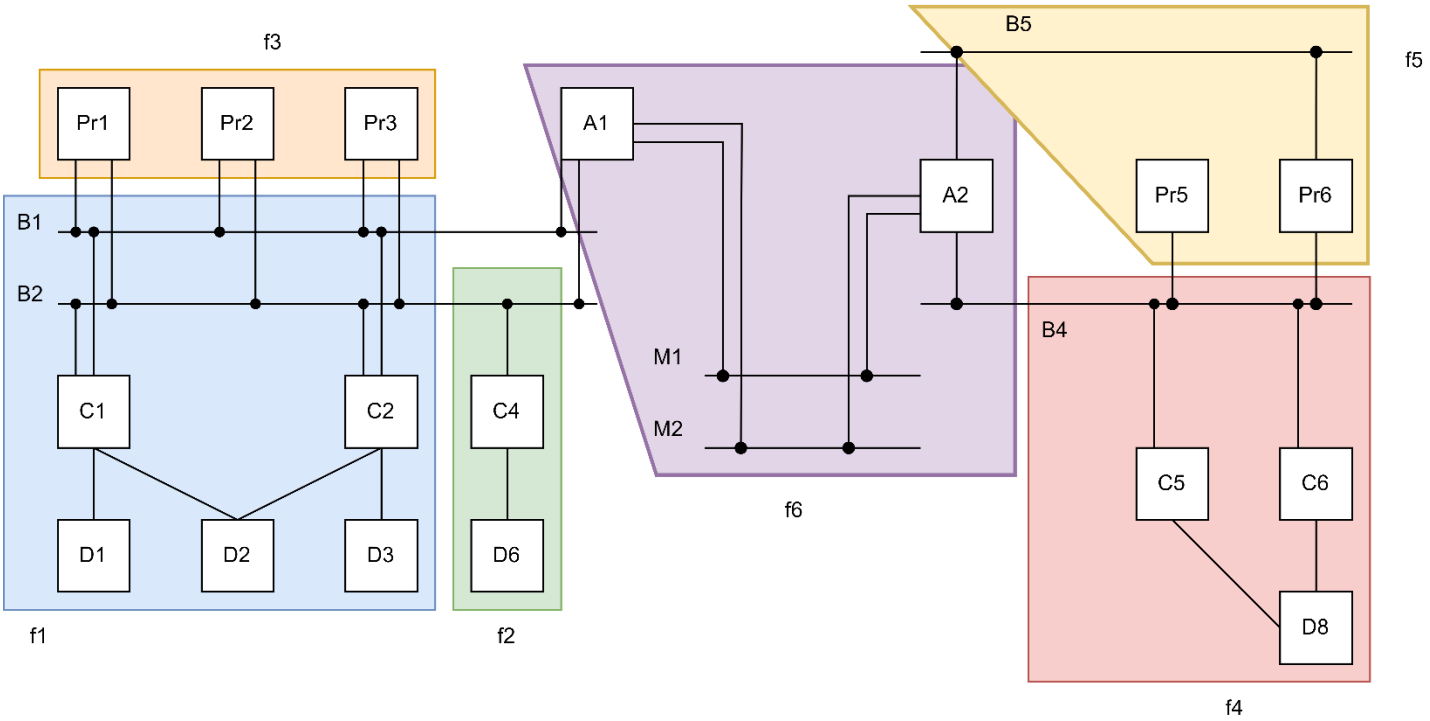


рис.1 Схема варіант 17

Складові логічної функції F f1,f2,f3,f4,f5,f6:

$$f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2)$$

$$f2 = d6 * c4 * b2$$

$$f3 = pr1 * pr2 * pr3$$

$$f4 = d8 * (c5 + c6) * b4$$

$$f5 = pr5 * pr6 * b5$$

$$f6 = a1 * a2 * (m1 + m2)$$

$$F = f1 * f2 * f3 * f4 * f5 * f6$$

Таблиця реконфігурації (Ln – номінальне навантаження, Lm – максимальне навантаження):

	Ln	Lm	Перерозподіли навантаження
pr1	50	80	[pr2 = 25, pr3 = 25]
pr2	50	80	[pr1 = 25, pr3 = 25]
pr3	50	80	[pr1 = 25, pr2 = 25]
pr5	30	60	відсутні
pr6	30	60	відсутні

Таблиця вказує перерозподіли:

Якщо pr1 відмовив фізично – 25 одиниць навантаження додаються до pr2, 25 одиниць навантаження додаються до pr3.

Якщо pr2 відмовив фізично – 25 одиниць навантаження додаються до pr1, 25 одиниць навантаження додаються до pr3.

Якщо pr3 відмовив фізично – 25 одиниць навантаження додаються до pr1, 25 одиниць навантаження додаються до pr2.

Для процесорів pr5, pr6 перерозподіли відсутні.

2. Опис програмного рішення

Для автоматизації розрахунків було створено програму мовою C++, що обчислює надійність системи. Посилання на github репозиторій: Bohdan628318ylypchenko/SchemeReliability: Scheme reliability lab by Bohdan Pylypchenko and Ihor Bezrukov.

Ключові особливості алгоритму та реалізації:

- Програма оброблює BCI можливі початкові вектори станів системи (для варіанту 17 кількість всіх можливих станів = $2^{23} = 8.388.608$). Для здійснення обчислень за прийнятний час програма використовує паралелізм: головний потік розподіляє вектори станів системи між потоками-обробниками згідно принципу “round robin” (подібно до того як карти розподіляються між гравцями у покері).
- Програма зберігає результати обчислень як набір .ssv файлів. У файлах зберігаються:
    - Початковий стан системи (F(sv1)).
    - Стан системи після реконфігурації (F(sv2)).
    - Імовірність виникнення стану sv1.
    - Значення «координат» вектору sv1.
    - Значення «координат» вектору sv2.
- Для більш глибокого аналізу надійності системи було створено скрипт мовою python (+ pandas). На основі .ssv файлів скрипт обчислює метрики:
    - Імовірність відмови для кожного елементу до реконфігурації та після реконфігурації: сума імовірностей всіх векторів станів, для яких схема відмовила, у яких даний елемент відмовив.
    - Кількість відмов для кожного елементу.
    - Імовірність роботоздатності / відмови системи (P, Q) в якості додаткової перевірки правильності розрахунків основної програми.
- Також скрипт перевіряє, чи виникли ситуації, коли F(sv1) = 1 але F(sv2) = 0. Якщо така ситуація відсутня, реконфігурація є консистентною. В противному випадку конфігурація є неконсистентною.

Схема загального алгоритму обчислення надійності, що реалізований програмою, наведена на рис. 2.

файл: alrorithm.ixx
клас: SchemeReliabilityCalculator

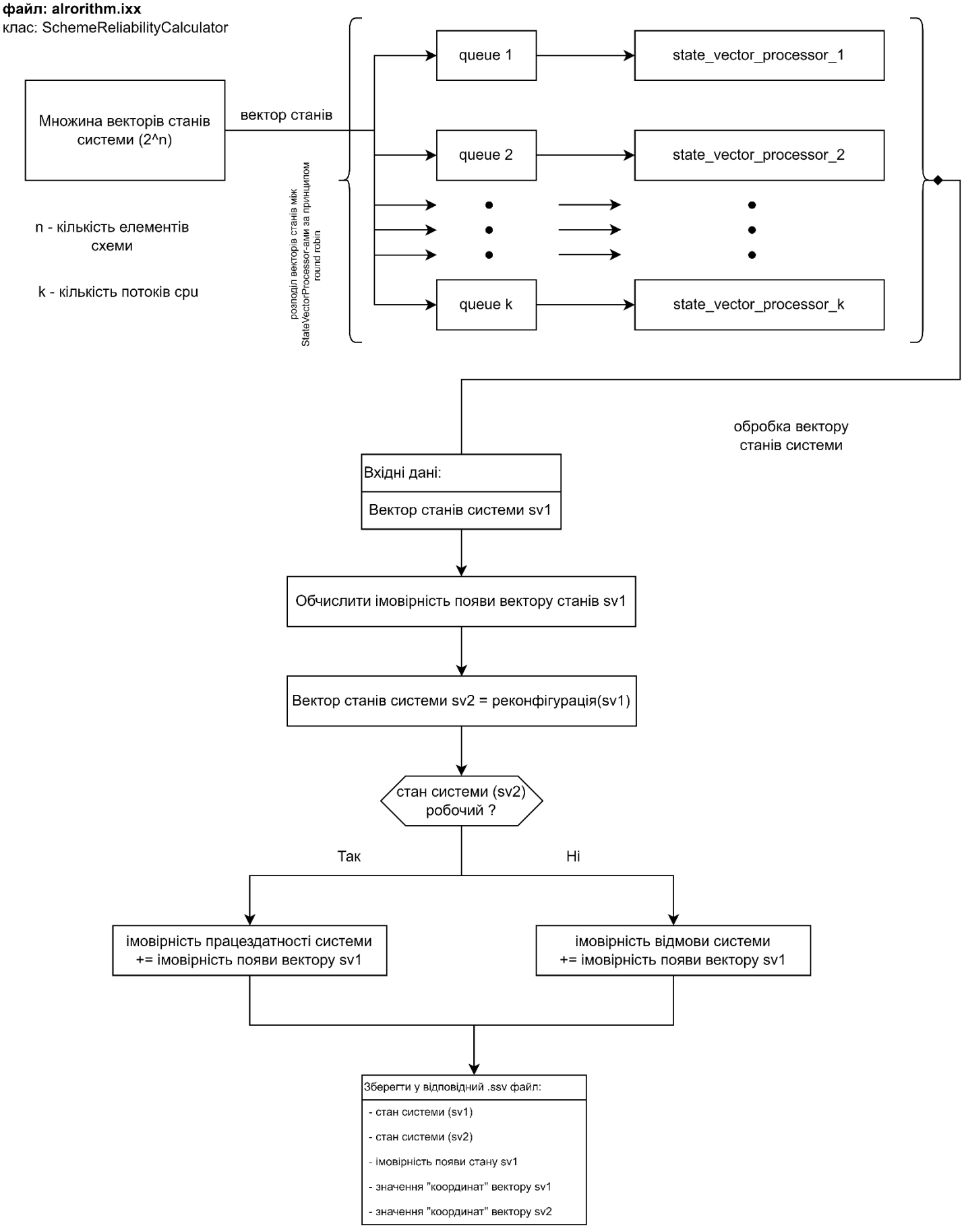


Рис. 2 Алгоритм обчислення надійності системи

У роботі реалізовано 2 алгоритми реконфігурації:

- Brute force (майже повний перебір всіх комбінацій перерозподілів).
- Жадібний алгоритм.

	Ln	Lm	Перерозподіли навантаження
pr1	50	100	[pr2 = 40, pr3 = 10], [pr2 = 30, pr3 = 20], [pr2 = 25, pr3 = 25], [pr2 = 20, pr3 = 30], [pr2 = 10, pr3 = 40], [pr2 = 50], [pr3 = 50]
pr2	50	100	[pr1 = 40, pr3 = 10], [pr1 = 30, pr3 = 20], [pr1 = 25, pr3 = 25], [pr1 = 20, pr3 = 30], [pr1 = 10, pr3 = 40], [pr1 = 50], [pr3 = 50]
pr3	50	100	[pr1 = 40, pr2 = 10], [pr1 = 30, pr2 = 20], [pr1 = 25, pr2 = 25], [pr1 = 20, pr2 = 30], [pr1 = 10, pr2 = 40], [pr1 = 50], [pr2 = 50]
pr5	30	60	[pr6 = 30], [pr1 = 20, pr2 = 20, pr3 = 20], [pr1 = 25, pr2 = 25], [pr2 = 25, pr3 = 25], [pr1 = 25, pr3 = 25], [pr1 = 30], [pr2 = 30], [pr3 = 30]
pr6	30	60	[pr5 = 30], [pr1 = 20, pr2 = 20, pr3 = 20], [pr1 = 25, pr2 = 25], [pr2 = 25, pr3 = 25], [pr1 = 25, pr3 = 25], [pr1 = 30], [pr2 = 30], [pr3 = 30]

Таблиця реконфігурацій задає набір можливих переходів (перерозподілів) навантажень для процесорів системи. Згідно таблиці вище, для процесорів pr1-3 маємо по 7 варіантів перерозподілів, для процесорів 5, 6 – по 8

Вважатимемо перерозподіл **можливим**, якщо всі процесори, на які відводиться навантаження, є робочими.

Вважатимемо перерозподіл **успішним**, якщо одночасно виконуються дві умови:

1. перерозподіл є можливим
2. навантаження на процесори перерозподілу менші за відповідні максимальні значення.

Нехай відмовили процесори 1, 3, 6. Тоді, згідно таблиці реконфігурації, існує 7 * 7 * 8 = 392 повних варіантів реконфігурації (комбінацій перерозподілів), 7 * 7 + 2 * 7 * 8 часткових реконфігурацій розмірності 2, 7 + 7 + 8 реконфігурацій розмірності 1. Разом 575 варіантів.

Ключові властивості brute force алгоритму:

- В основі алгоритму лежить обхід дерева вглибину за допомогою рекурсії.
- Обхід дерева зупиняється передчасно, якщо знайдено реконфігурацію, для якої F(sv2) = 1.
- Якщо не існує такої реконфігурації, що F(sv2) = 1, алгоритм обиратиме реконфігурацію із максимальною кількістю робочих елементів.
- Якщо перерозподіл, що розглядається, не є можливим (внаслідок виборів на попередніх рівнях дерева), алгоритм переходить на наступний рівень дерева, «перестрибуючи» перерозподіли поточного рівня. Такі «стрибки» є причиною появи «часткових» реконфігурацій. При цьому після того, як алгоритм дійде до останнього рівня дерева, обхід повернеться до пропущеного рівня дерева і перейде до інших перерозподілів пропущеного рівня.
- В найгіршому випадку алгоритм перебиратиме всі варіанти реконфігурації.

Схема brute force алгоритму зображена на рис. 3:

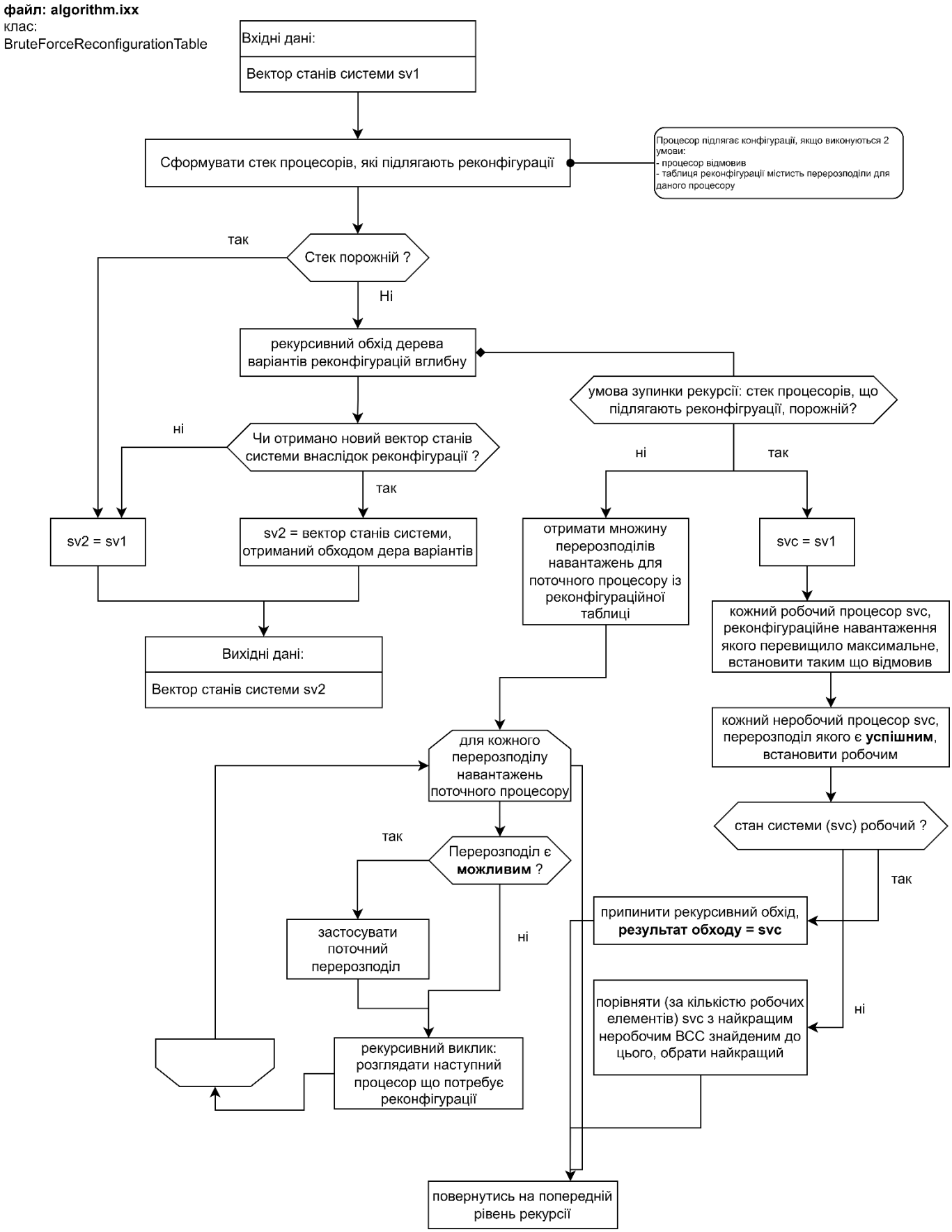


Рис. 3 Схема brute force алгоритму

Очевидно, алгоритм повного обходу дерева є повільним. З’являється потреба у алгоритмі, який, можливо, не знаходитиме найкращий варіант реконфігурації, але знаходитиме реконфігурацію задовільної якості за набагато менший час.

Введемо оцінку навантаження J:

$$J(l) = \sum_{i=1}^{n} \left(\frac{l_i}{l_i^{(m)}} + \theta\left(\frac{l_i}{l_i^{(m)}} - 1 \right) * K \right)$$

де:

n — кількість елементів системи

l – вектор навантаження

$l^{(m)}$ — вектор максимальних навантажень

$\theta(x)$ — функція Гевісайда:

$$\theta(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

K — штрафний коефіцієнт. В роботі значення K рівне 10^6.

Чим меншою є оцінка, тим кращим вважається навантаження.

На основі оцінки J було створено жадібний алгоритм:

файл: algorithm.ixx
клас: GreedyReconfigurationTable

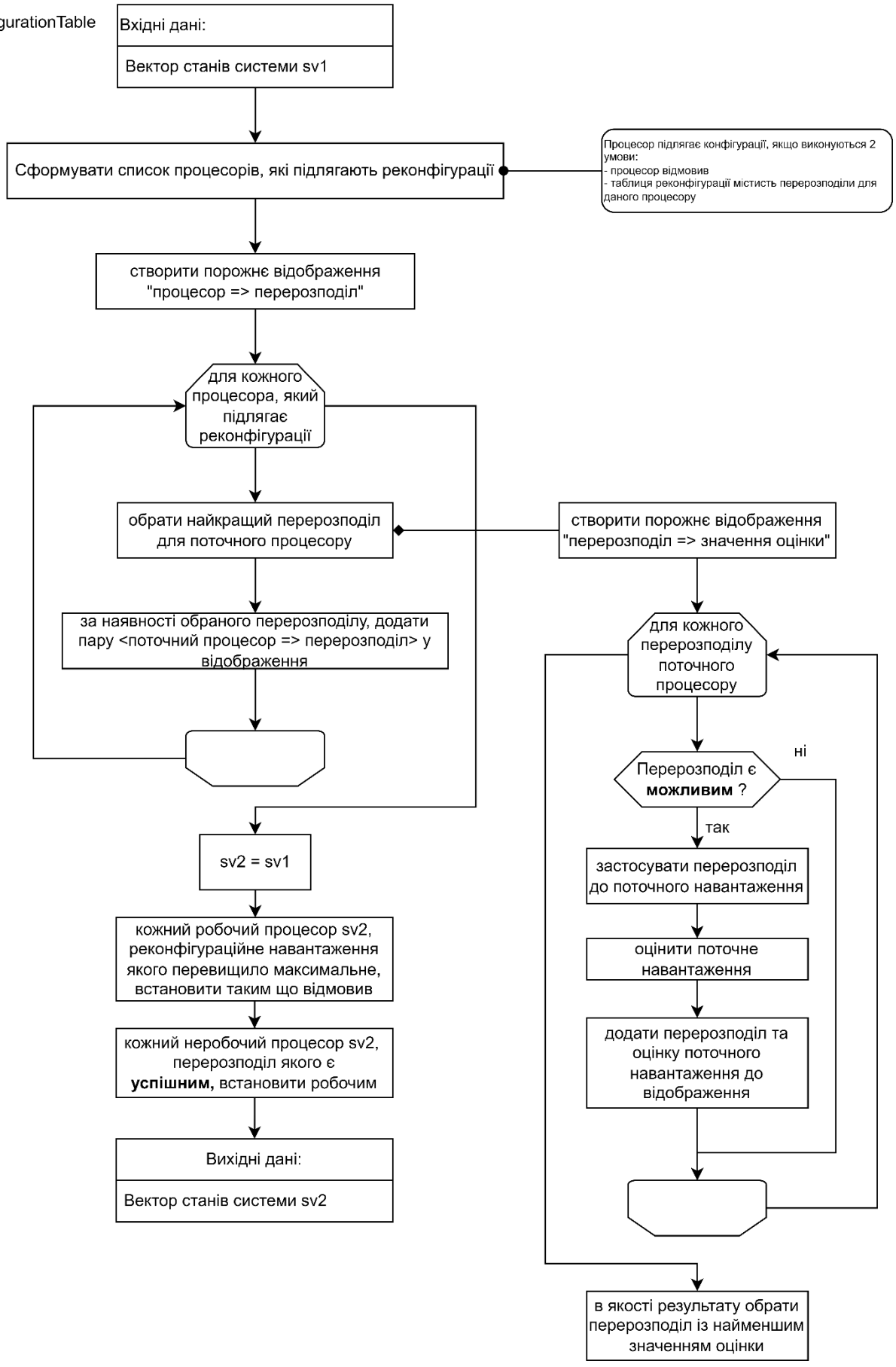


Рис.4 Схема жадібного алгоритму

Для кожного процесору, що потребує реконфігурації, алгоритм оцінює всі **можливі** (можливі у сенсі вищенаведеного визначення) перерозподіли, обирає найкращий. Таким чином для відмов pr1, pr3, pr6 замість 575 варіантів реконфігурацій жадібному алгоритму потрібно оцінити 7 + 7 + 8 = 22 перерозподіли. Іншими словами, жадібний алгоритм дозволив «позбутись» вибухової складності, спричиненої основним правилом комбінаторики.

3. Дослідження надійності

Через великі об’єми обчислень лабораторна робота виконувалась на AWS EC2, тип віртуальної машини – c7i.8xlarge (16 ядер / 32 потоки, 64 GB ddr5):

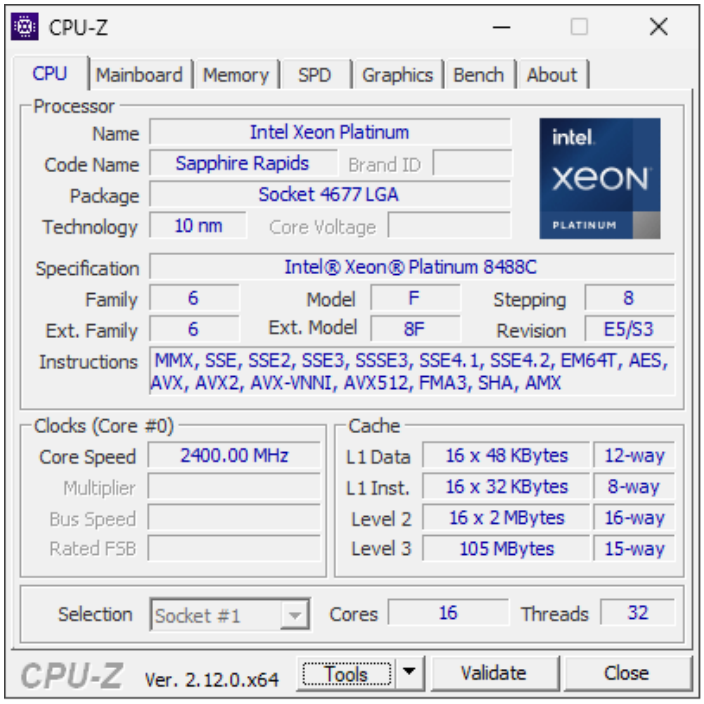


Рис. 5 Процесор, використаний для обчислень

Повний перелік характеристик vm можна переглянути у файлі ec2-specs.txt у репозиторії.

Вивід програми для всіх модифікацій схем (одиниця виміру часу – секунда):

```
PS C:\Users\Administrator\SchemeReliability\x64\Release> .\sr-research.exe


=== original s23 ===


Scheme type = greedy
time = 3
path = s23-original-greedy
sp = 0.9990208569798796, sq = 0.000979143020119051
state count = 8388608


Scheme type = brute
time = 3
path = s23-original-brute
sp = 0.9990208569798796, sq = 0.000979143020119051
state count = 8388608


=== s23 with rt (7 7 7 8 8) ===
```

```
Scheme type = greedy
time = 2
path = s23-77788-greedy
sp = 0.9992607082975051, sq = 0.0007392917024915616
state count = 8388608

Scheme type = brute
time = 18
path = s23-77788-brute
sp = 0.9992607082975051, sq = 0.0007392917024915616
state count = 8388608

=== s23 with rt and modified connections ===

Scheme type = greedy
time = 3
path = s23-77788-modified-connections-greedy
sp = 0.999737721521635, sq = 0.0002622784783607316
state count = 8388608

Scheme type = brute
time = 13
path = s23-77788-modified-connections-brute
sp = 0.999737721521635, sq = 0.0002622784783607316
state count = 8388608

=== s24 with d9 right ===

Scheme type = greedy
time = 1
path = s24-d9-right-greedy
sp = 0.9997597157515095, sq = 0.00024028424848725232
state count = 16777216

Scheme type = brute
time = 35
path = s24-d9-right-brute
sp = 0.9997597157515095, sq = 0.00024028424848725232
state count = 16777216

=== s25 with d9 d10 right ===
```

```
Scheme type = greedy
time = 8
path = s25-d9-d10-right-greedy
sp = 0.9997597162353798, sq = 0.00024028376461419955
state count = 33554432

Scheme type = brute
time = 61
path = s25-d9-d10-right-brute
sp = 0.9997597162353798, sq = 0.00024028376461419955
state count = 33554432

=== s27 with d9 d10 c7 right c8 left ===

Scheme type = greedy
time = 77
path = s27-d9-d10-c7-right-c8-left-greedy
sp = 0.9997598842949585, sq = 0.00024011570500589362
state count = 134217728

Scheme type = brute
time = 220
path = s27-d9-d10-c7-right-c8-left-brute
sp = 0.9997598842949585, sq = 0.00024011570500589362
state count = 134217728

=== s29 with d9 d10 c7 right c8 left a4 ===

Scheme type = greedy
time = 397
path = s29-d9-d10-c7-right-c8-left-a4-greedy
sp = 0.999999841063663, sq = 1.5893623277066473e-07
state count = 536870912

Scheme type = brute
time = 1022
path = s29-d9-d10-c7-right-c8-left-a4-brute
sp = 0.999999841063663, sq = 1.5893623277066473e-07
state count = 536870912
PS C:\Users\Administrator\SchemeReliability\x64\Release>
```

3.1 Початкова схема

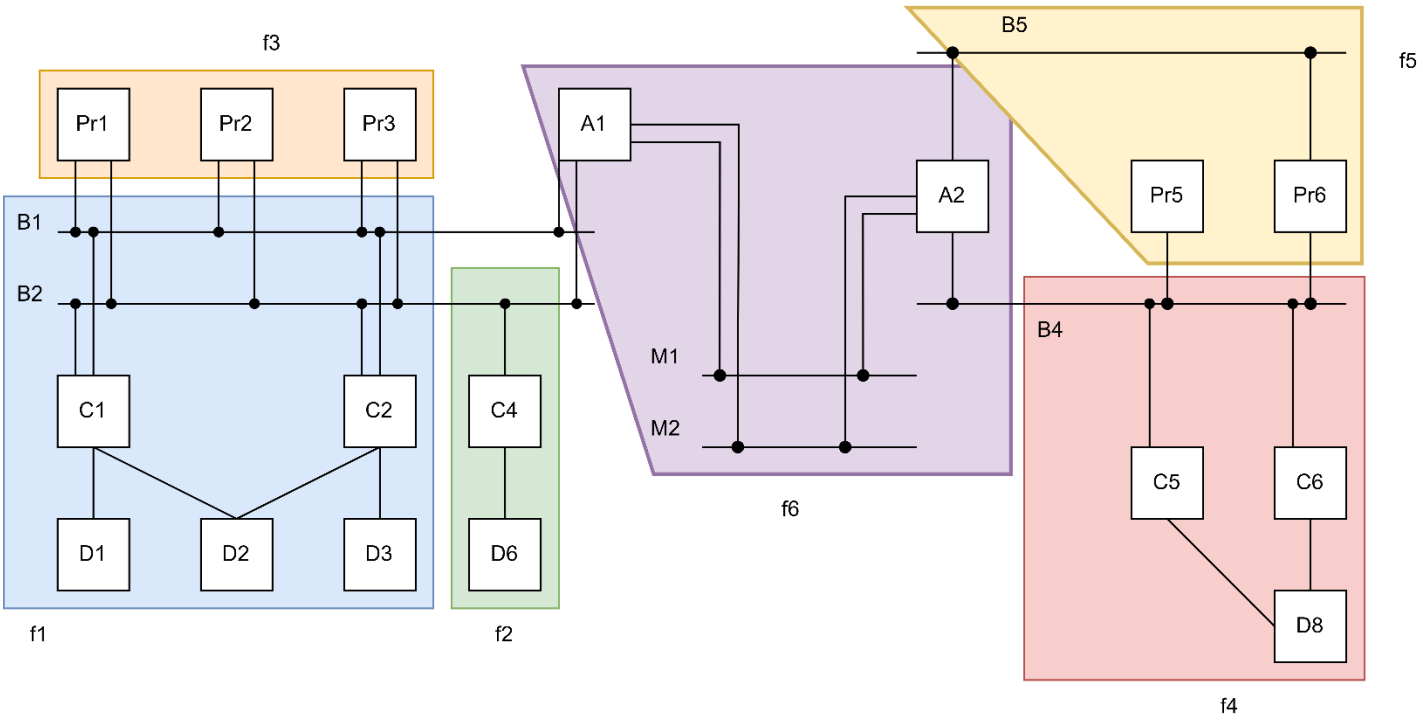


Рис. 6 Оригінальна схема

Опис логічної функції:

$$f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2)$$

$$f2 = d6 * c4 * b2$$

$$f3 = pr1 * pr2 * pr3$$

$$f4 = d8 * (c5 + c6) * b4$$

$$f5 = pr5 * pr6 * b5$$

$$f6 = a1 * a2 * (m1 + m2)$$

$$F = f1 * f2 * f3 * f4 * f5 * f6$$

Таблиця реконфігурації:

	Ln	Lm	Перерозподіли навантаження
pr1	50	80	[pr2 = 25, pr3 = 25]
pr2	50	80	[pr1 = 25, pr3 = 25]
pr3	50	80	[pr1 = 25, pr2 = 25]
pr5	30	60	відсутні
pr6	30	60	відсутні

Результати виконання програми:

```
=== original s23 ===

Scheme type = greedy
time = 3
path = s23-original-greedy
sp = 0.9990208569798796, sq = 0.000979143020119051
state count = 8388608

Scheme type = brute
time = 3
path = s23-original-brute
sp = 0.9990208569798796, sq = 0.000979143020119051
state count = 8388608
```

Метрики (жадібний алгоритм ліворуч, повний обхід дерева - праворуч):

```
In [6]: process_sample_with_fail_count_per_123("s23-original-greedy")

=== sum probabilities ===
p = 0.9990208569798807
q = 0.0009791430201191219
p + q = 0.9999999999999999
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr2: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr3: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr4: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
pr5: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 2.00805635e-08, sv2 = 1.46871453e-08
b2: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b4: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b5: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
c1: sv1 = 7.16675097e-07, sv2 = 5.69315417e-07
c2: sv1 = 7.16675097e-07, sv2 = 5.69315417e-07
c4: sv1 = 4.10000000e-04, sv2 = 4.10000000e-04
c5: sv1 = 7.16674899e-07, sv2 = 5.69315219e-07
c6: sv1 = 7.16674899e-07, sv2 = 5.69315219e-07
d1: sv1 = 2.94517019e-08, sv2 = 2.15413552e-08
d2: sv1 = 2.94518999e-08, sv2 = 2.15415534e-08
d3: sv1 = 2.94517019e-08, sv2 = 2.15413552e-08
d6: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 6.11313451e-07, sv2 = 4.81917997e-07
m2: sv1 = 6.11313451e-07, sv2 = 4.81917997e-07
=== sum fail count(sv1 | sv2) per pr1 * pr2 * pr3 state set ===
case 0: sv1 = 1048576; sv2 = 4194304
case 1: sv1 = 1048576; sv2 = 0
case 2: sv1 = 1048576; sv2 = 0
case 3: sv1 = 1048576; sv2 = 1048576
case 4: sv1 = 1048576; sv2 = 0
case 5: sv1 = 1048576; sv2 = 1048576
case 6: sv1 = 1048576; sv2 = 1048576
case 7: sv1 = 1048576; sv2 = 1048576
```

```
In [7]: process_sample_with_fail_count_per_123("s23-original-brute")

=== sum probabilities ===
p = 0.9990208569798807
q = 0.0009791430201191219
p + q = 0.9999999999999999
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr2: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr3: sv1 = 1.20000000e-04, sv2 = 2.87982720e-08
pr4: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
pr5: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 2.00805635e-08, sv2 = 1.46871453e-08
b2: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b4: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b5: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
c1: sv1 = 7.16675097e-07, sv2 = 5.69315417e-07
c2: sv1 = 7.16675097e-07, sv2 = 5.69315417e-07
c4: sv1 = 4.10000000e-04, sv2 = 4.10000000e-04
c5: sv1 = 7.16674899e-07, sv2 = 5.69315219e-07
c6: sv1 = 7.16674899e-07, sv2 = 5.69315219e-07
d1: sv1 = 2.94517019e-08, sv2 = 2.15413552e-08
d2: sv1 = 2.94518999e-08, sv2 = 2.15415534e-08
d3: sv1 = 2.94517019e-08, sv2 = 2.15413552e-08
d6: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 6.11313451e-07, sv2 = 4.81917997e-07
m2: sv1 = 6.11313451e-07, sv2 = 4.81917997e-07
=== sum fail count(sv1 | sv2) per pr1 * pr2 * pr3 state set ===
case 0: sv1 = 1048576; sv2 = 4194304
case 1: sv1 = 1048576; sv2 = 0
case 2: sv1 = 1048576; sv2 = 0
case 3: sv1 = 1048576; sv2 = 1048576
case 4: sv1 = 1048576; sv2 = 0
case 5: sv1 = 1048576; sv2 = 1048576
case 6: sv1 = 1048576; sv2 = 1048576
case 7: sv1 = 1048576; sv2 = 1048576
```

1. Результати алгоритмів є однаковими.
2. Маємо надійність, що є меншою за 0.9999. Поставимо за мету досягнути надійність, більшу або рівну 0.9999.
3. Реконфігурація очікувано зменшила імовірність відмови процесорів 1, 2, 3.
4. Імовірність відмови процесорів 5 і 6 очікувано не змінилась – перерозподіли для цих процесорів відсутні у таблиці реконфігурації.

5. Час виконання обох алгоритмів – 3 секунди. Схоже розмірність обчислень ($2^{23} = 8388608$) і мала кількість варіантів реконфігурації для початкової схеми є недостатньо великою щоб різниця між алгоритмами була помітна.

3.2 Модифікація 1: 7-7-7-8-8

Модифікуємо таблицю реконфігурації (додамо перерозподіли, збільшимо максимальне навантаження для процесорів 1, 2, 3 із 80 до 100):

	Ln	Lm	Перерозподіли навантаження
pr1	50	100	[pr2 = 50], [pr3 = 50], [pr2 = 25, pr3 = 25], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr2	50	100	[pr1 = 50], [pr3 = 50], [pr1 = 25, pr3 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr3	50	100	[pr1 = 50], [pr2 = 50], [pr1 = 25, pr2 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30]
pr5	30	60	[pr6 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]
pr6	30	60	[pr5 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]

Інші параметри схеми залишимо без змін.

Результати виконання програми:

```
17  === s23 with rt (7 7 7 8 8) ===
18
19  Scheme type = greedy
20  time = 2
21  path = s23-77788-greedy
22  sp = 0.9992607082975051, sq = 0.0007392917024915616
23  state count = 8388608
24
25  Scheme type = brute
26  time = 18
27  path = s23-77788-brute
28  sp = 0.9992607082975051, sq = 0.0007392917024915616
29  state count = 8388608
```

Метрики (жадібний алгоритм ліворуч, повний обхід дерева - праворуч):

```
In [8]: process_sample_with_fail_count_per_123("s23-77788-greedy")
```

```
=== sum probabilities ===
p = 0.9992607082975082
q = 0.0007392917024916249
p + q = 0.9999999999999999
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 2.00805635e-08, sv2 = 1.10893755e-08
b2: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b4: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b5: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
c1: sv1 = 7.16675097e-07, sv2 = 4.71016679e-07
c2: sv1 = 7.16675097e-07, sv2 = 4.71016679e-07
c4: sv1 = 4.10000000e-04, sv2 = 4.10000000e-04
c5: sv1 = 7.16674899e-07, sv2 = 4.71016481e-07
c6: sv1 = 7.16674899e-07, sv2 = 4.71016481e-07
d1: sv1 = 2.94517019e-08, sv2 = 1.62646263e-08
d2: sv1 = 2.94518999e-08, sv2 = 1.62648245e-08
d3: sv1 = 2.94517019e-08, sv2 = 1.62646263e-08
d6: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 6.11313451e-07, sv2 = 3.95602596e-07
m2: sv1 = 6.11313451e-07, sv2 = 3.95602596e-07
=== sum fail count(sv1 | sv2) per pr1 * pr2 * pr3 state set ===
case 0: sv1 = 1048576; sv2 = 4194304
case 1: sv1 = 1048576; sv2 = 0
case 2: sv1 = 1048576; sv2 = 0
case 3: sv1 = 1048576; sv2 = 524288
case 4: sv1 = 1048576; sv2 = 0
case 5: sv1 = 1048576; sv2 = 524288
case 6: sv1 = 1048576; sv2 = 524288
case 7: sv1 = 1048576; sv2 = 2621440
```

```
In [9]: process_sample_with_fail_count_per_123("s23-77788-brute")
```

```
=== sum probabilities ===
p = 0.9992607082975082
q = 0.0007392917024916249
p + q = 0.9999999999999999
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 2.00805635e-08, sv2 = 1.10893755e-08
b2: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b4: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
b5: sv1 = 1.50000000e-05, sv2 = 1.50000000e-05
c1: sv1 = 7.16675097e-07, sv2 = 4.71016679e-07
c2: sv1 = 7.16675097e-07, sv2 = 4.71016679e-07
c4: sv1 = 4.10000000e-04, sv2 = 4.10000000e-04
c5: sv1 = 7.16674899e-07, sv2 = 4.71016481e-07
c6: sv1 = 7.16674899e-07, sv2 = 4.71016481e-07
d1: sv1 = 2.94517019e-08, sv2 = 1.62646263e-08
d2: sv1 = 2.94518999e-08, sv2 = 1.62648245e-08
d3: sv1 = 2.94517019e-08, sv2 = 1.62646263e-08
d6: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 6.11313451e-07, sv2 = 3.95602596e-07
m2: sv1 = 6.11313451e-07, sv2 = 3.95602596e-07
=== sum fail count(sv1 | sv2) per pr1 * pr2 * pr3 state set ===
case 0: sv1 = 1048576; sv2 = 4980736
case 1: sv1 = 1048576; sv2 = 786432
case 2: sv1 = 1048576; sv2 = 1572864
case 3: sv1 = 1048576; sv2 = 0
case 4: sv1 = 1048576; sv2 = 0
case 5: sv1 = 1048576; sv2 = 0
case 6: sv1 = 1048576; sv2 = 0
case 7: sv1 = 1048576; sv2 = 1048576
```

1. Імовірність безвідмовної роботи системи збільшилась: 0.9992 проти 0.9990.
2. Результати реконфігурації відрізняються: алгоритм повного обходу знаходить комбінації перерозподілів, у яких кількість робочих процесорів є більшою у порівнянні з реконфігураціями, знайденими жадібним алгоритмом. При цьому різниця є несуттєвою, адже не вплинула на кінцеве значення імовірності безвідмовної роботи системи.
3. Помітна різниця у складності алгоритмів: жадібний алгоритм закінчив обчислення за 2 секунди, для повного обходу знадобилось 18 секунд.

3.3 Модифікація 2: з’єднання елементів

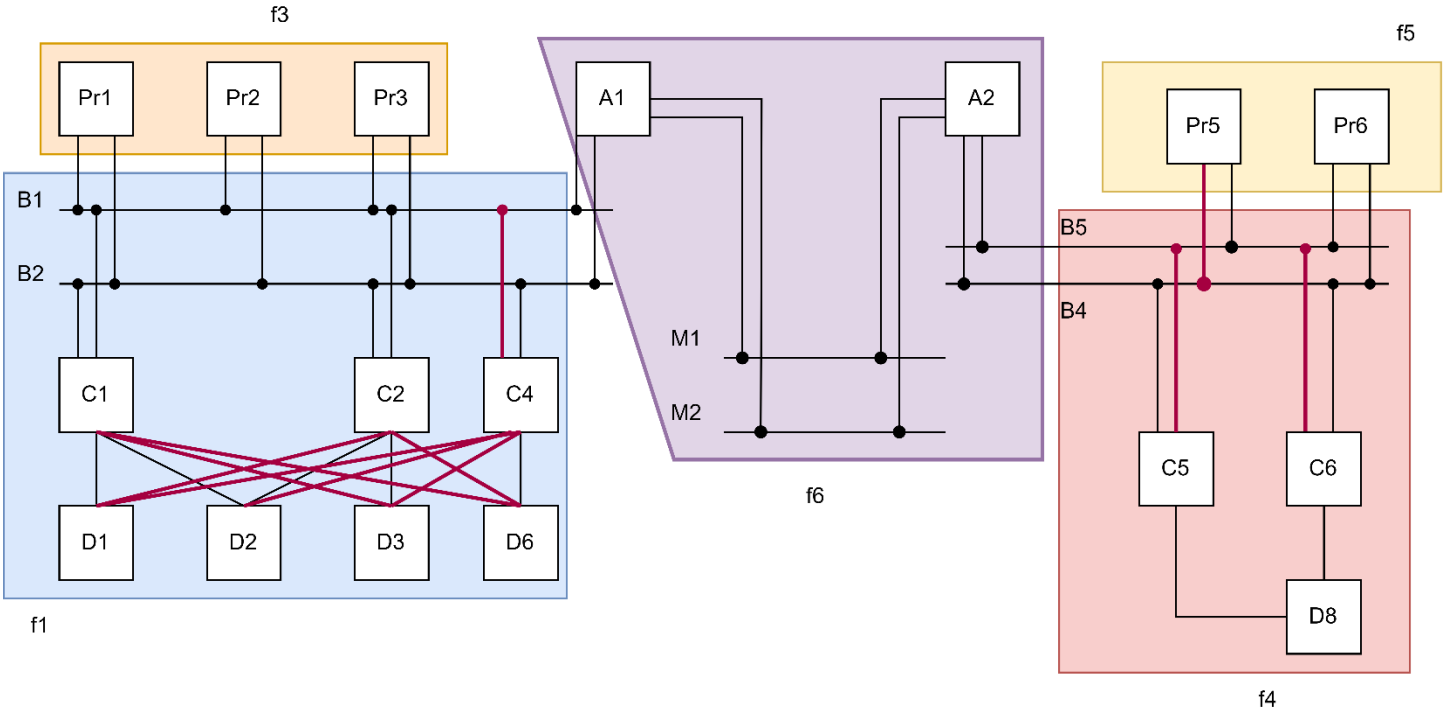


Рис. 7 Схема з модифікованими з’єднаннями

Утворимо додаткові з’єднання (на рисунку позначені червоним кольором).

Внаслідок нових з’єднань логічна функція матиме вид:

$$f1 = (d1 + d2 + d3 + d4) * (c1 + c2 + c4) * (b1 + b2)$$

$$f3 = pr1 * pr2 * pr3$$

$$f4 = d8 * (c5 + c6) * (b4 + b5)$$

$$f5 = pr5 * pr6$$

$$f6 = a1 * a2 * (m1 + m2)$$

$$F = f1 * f3 * f4 * f5 * f6$$

Інші параметри залишимо без змін.

Опис нової логічної функції:

```
.scheme_function = [](const StateVectorDto<all_count, processor_count>& sv)
{
    span<bool> s = sv.all;

    bool f1 = (s[16] + s[17] + s[18] + s[19]) * (s[11] + s[12] + s[13]) * (s[7] + s[8]);
    bool f3 = s[0] * s[1] * s[2];
    bool f4 = s[20] * (s[14] + s[15]) * (s[9] + s[10]);
    bool f5 = s[3] * s[4];
    bool f6 = s[5] * s[6] * (s[21] + s[22]);
```

```
        return f1 * f3 * f4 * f5 * f6;
    }
```

Результати виконання програми:

```
=== s23 with rt and modified connections ===

Scheme type = greedy
time = 3
path = s23-77788-modified-connections-greedy
sp = 0.999737721521635, sq = 0.0002622784783607316
state count = 8388608

Scheme type = brute
time = 13
path = s23-77788-modified-connections-brute
sp = 0.999737721521635, sq = 0.0002622784783607316
state count = 8388608
```

Метрики (жадібний алгоритм ліворуч, повний обхід дерева – праворуч):

```
In [10]: process_sample_with_del("s23-77788-modified-connections-greedy")
=== sum probabilities ===
p = 0.999737721521639
q = 0.0002622784783607359
p + q = 0.9999999999999997
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b2: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b4: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b5: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
c1: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c2: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c4: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c5: sv1 = 5.21296899e-07, sv2 = 2.75521212e-07
c6: sv1 = 5.21296899e-07, sv2 = 2.75521212e-07
d1: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d2: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d3: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d6: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 4.39753462e-07, sv2 = 2.23939634e-07
m2: sv1 = 4.39753462e-07, sv2 = 2.23939634e-07
```

```
In [11]: process_sample_with_del("s23-77788-modified-connections-brute")
=== sum probabilities ===
p = 0.999737721521639
q = 0.0002622784783607359
p + q = 0.9999999999999997
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b2: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b4: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
b5: sv1 = 1.31544599e-08, sv2 = 4.15911479e-09
c1: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c2: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c4: sv1 = 3.53479464e-07, sv2 = 1.07603051e-07
c5: sv1 = 5.21296899e-07, sv2 = 2.75521212e-07
c6: sv1 = 5.21296899e-07, sv2 = 2.75521212e-07
d1: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d2: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d3: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d6: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
d8: sv1 = 2.20000000e-05, sv2 = 2.20000000e-05
m1: sv1 = 4.39753462e-07, sv2 = 2.23939634e-07
m2: sv1 = 4.39753462e-07, sv2 = 2.23939634e-07
```

1. Імовірність безвідмовної роботи збільшилась: 0.9997 проти 0.9992.
2. Результати реконфігурації відрізняються: алгоритм повного обходу знаходить комбінації перерозподілів, у яких кількість робочих процесорів є більшою у порівнянні з реконфігураціями, знайденими жадібним алгоритмом. При цьому різниця є несуттєвою, адже не вплинула на кінцеве значення імовірності безвідмовної роботи системи.
3. Жадібний алгоритм знову демонструє перевагу у швидкодії: 2 секунди проти 13 секунд.

3.4 Модифікація 3: 24 елементи (d9)

Судячи із обчислених імовірностей відмов кожного елементу, реконфігурація успішно компенсує фізичні відмови процесорів, головним джерелом відмов є контролери. Тому для подальшого збільшення надійності необхідно дублювати існуючі методи паралельним з’єднанням.

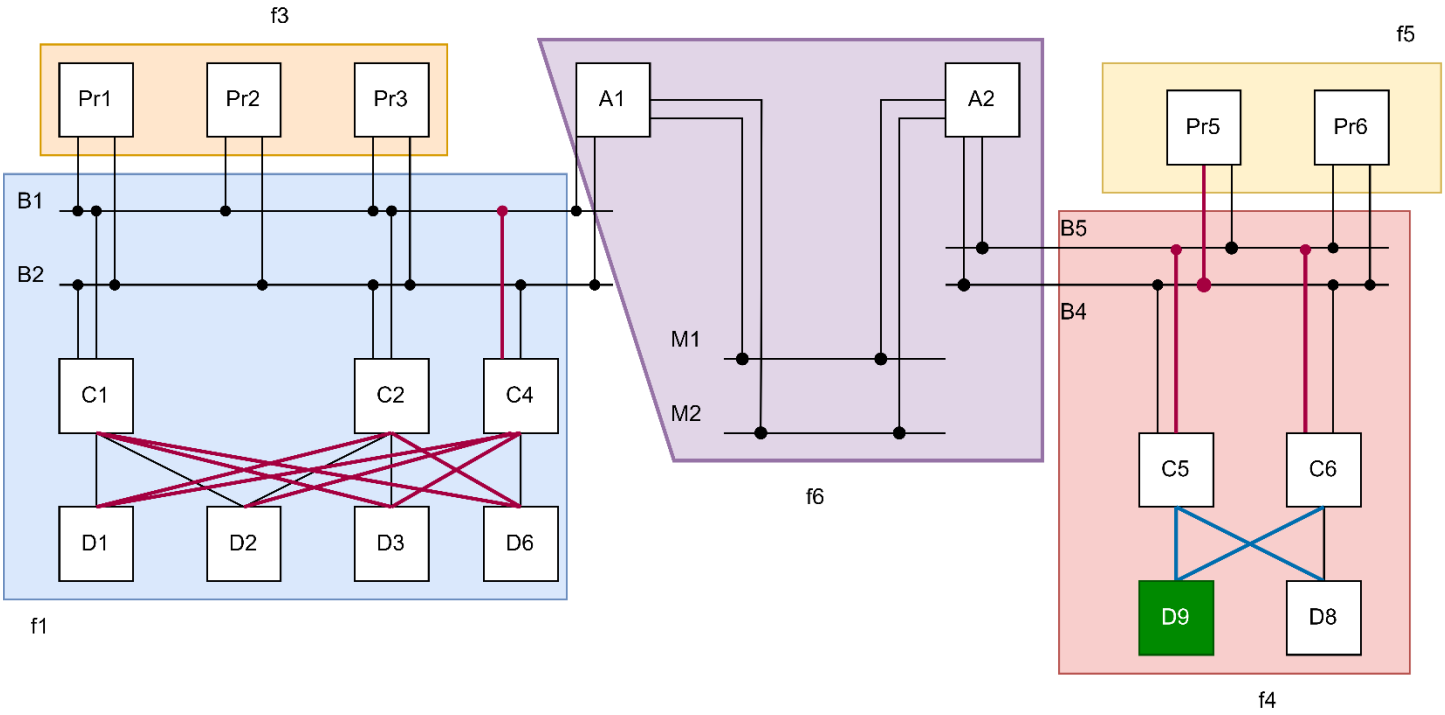


Рис. 8 Схема з новим елементом d9 (всього 24 елементи)

Додавання нового елементу d9 відобразиться на виді логічної функції:

$f4 = (d8 + d9) * (c5 + c6) * (b4 + b5)$

Інші параметри залишимо без змін.

Результати роботи програми:

```
45   === s24 with d9 right ===
46
47   Scheme type = greedy
48   time = 1
49   path = s24-d9-right-greedy
50   sp = 0.9997597157515095, sq = 0.00024028424848725232
51   state count = 16777216
52
53   Scheme type = brute
54   time = 35
55   path = s24-d9-right-brute
56   sp = 0.9997597157515095, sq = 0.00024028424848725232
57   state count = 16777216
```

Метрики:

```
In [12]: process_sample_with_del("s24-d9-right-greedy")
```

```
=== sum probabilities ===
p = 0.9997597157515122
q = 0.0002402842484872624
p + q = 0.9999999999999994
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b2: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b4: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b5: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
c1: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c2: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c4: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c5: sv1 = 5.12288368e-07, sv2 = 2.66507274e-07
c6: sv1 = 5.12288368e-07, sv2 = 2.66507274e-07
d1: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d2: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d3: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d6: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d8: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
m1: sv1 = 4.31843137e-07, sv2 = 2.16024561e-07
m2: sv1 = 4.31843137e-07, sv2 = 2.16024561e-07
d9: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
```

```
In [13]: process_sample_with_del("s24-d9-right-brute")
```

```
=== sum probabilities ===
p = 0.9997597157515122
q = 0.0002402842484872624
p + q = 0.9999999999999994
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b2: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b4: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
b5: sv1 = 1.28247493e-08, sv2 = 3.82920629e-09
c1: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c2: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c4: sv1 = 3.44467241e-07, sv2 = 9.85854181e-08
c5: sv1 = 5.12288368e-07, sv2 = 2.66507274e-07
c6: sv1 = 5.12288368e-07, sv2 = 2.66507274e-07
d1: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d2: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d3: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d6: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d8: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
m1: sv1 = 4.31843137e-07, sv2 = 2.16024561e-07
m2: sv1 = 4.31843137e-07, sv2 = 2.16024561e-07
d9: sv1 = 1.89634973e-08, sv2 = 5.77012652e-09
```

1. Надійність незначно збільшилась: 0.99975 проти 0.99973
2. Бачимо суттєву перевагу швидкодії жадібного алгоритму: 1 секунда проти 35. При цьому кінцеве значення надійності є однаковим для обох алгоритмів.

3.5 Модифікація 4: 25 елементів (d10)

У лівій частині схеми кількість контролерів типу d є більшою за кількість контролерів типу c на 1. Додамо ще один контролер типу d (d10), щоб права частина схеми також мала різницю між кількістю контролерів c і d рівну 1:

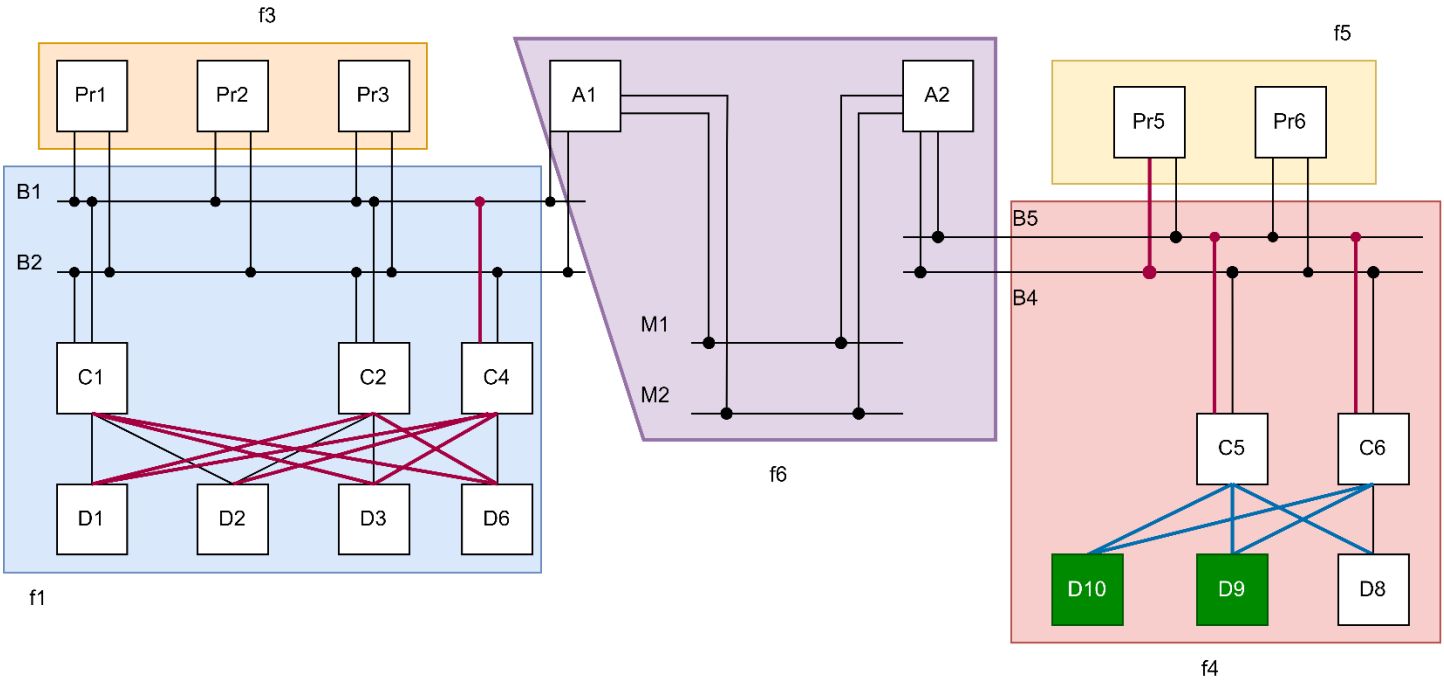


Рис. 9 Схема з новим елементом d10 (всього 25 елементів)

Додавання нового елементу d10 відобразиться на виді логічної функції:

$f4 = (d8 + d9 + d10) * (c5 + c6) * (b4 + b5)$

Інші параметри залишимо без змін.

Результати роботи програми:

```
59   === s25 with d9 d10 right ===
60
61   Scheme type = greedy
62   time = 8
63   path = s25-d9-d10-right-greedy
64   sp = 0.9997597162353798, sq = 0.00024028376461419955
65   state count = 33554432
66
67   Scheme type = brute
68   time = 61
69   path = s25-d9-d10-right-brute
70   sp = 0.9997597162353798, sq = 0.00024028376461419955
71   state count = 33554432
```

Метрики:

```
In [14]: process_sample_with_del("s25-d9-d10-right-greedy")
```

```
=== sum probabilities ===
p = 0.999759716235385
q = 0.00024028376461420454
p + q = 0.9999999999999991
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b2: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b4: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b5: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
c1: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c2: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c4: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c5: sv1 = 5.12288170e-07, sv2 = 2.66507076e-07
c6: sv1 = 5.12288170e-07, sv2 = 2.66507076e-07
d1: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d2: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d3: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d6: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d8: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
m1: sv1 = 4.31842963e-07, sv2 = 2.16024386e-07
m2: sv1 = 4.31842963e-07, sv2 = 2.16024386e-07
d9: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d10: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
```

```
In [15]: process_sample_with_del("s25-d9-d10-right-brute")
```

```
=== sum probabilities ===
p = 0.999759716235385
q = 0.00024028376461420454
p + q = 0.9999999999999991
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b2: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b4: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
b5: sv1 = 1.28247421e-08, sv2 = 3.82919903e-09
c1: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c2: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c4: sv1 = 3.44467043e-07, sv2 = 9.85852197e-08
c5: sv1 = 5.12288170e-07, sv2 = 2.66507076e-07
c6: sv1 = 5.12288170e-07, sv2 = 2.66507076e-07
d1: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d2: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d3: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d6: sv1 = 1.84799038e-08, sv2 = 5.28624282e-09
d8: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
m1: sv1 = 4.31842963e-07, sv2 = 2.16024386e-07
m2: sv1 = 4.31842963e-07, sv2 = 2.16024386e-07
d9: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
d10: sv1 = 1.84799145e-08, sv2 = 5.28625347e-09
```

1. Надійність майже не підвищилась: 0.9997597162**3** проти 0.9997597157**5**.
2. Знову спостерігаємо перевагу швидкодії жадібного алгоритму: 8 секунд проти 61. При цьому кінцеве значення надійності є однаковим для обох алгоритмів.

3.6 Модифікація 5: 27 елементів (c7, c8)

Додамо контролери c7 та c8 до правої та лівої частини схеми:

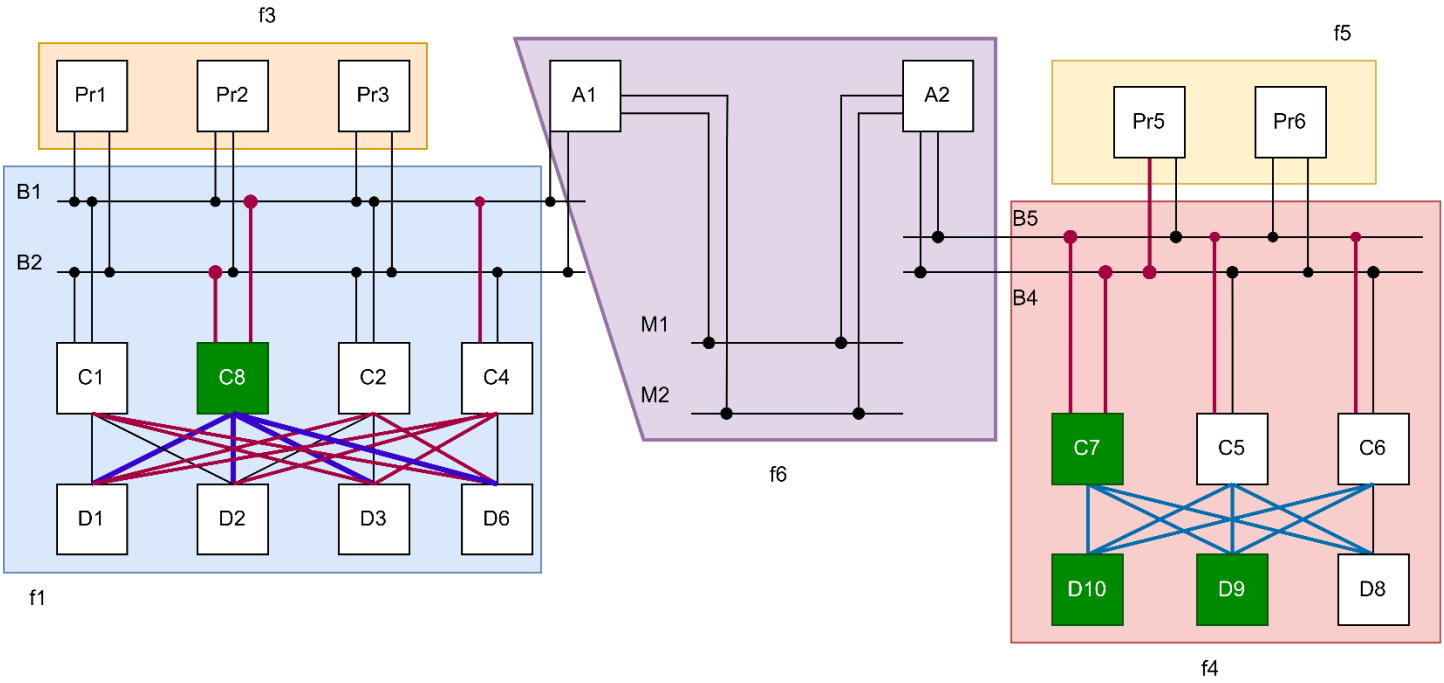


Рис. 10 Схема з елементами c7 та c8 (всього 27 елементів).

Зміни функцій $f1$ та $f4$:

$$f1 = (d1 + d2 + d3 + d4) * (c1 + c8 + c2 + c4) * (b1 + b2)$$

$$f4 = (d8 + d9 + d10) * (c5 + c6) * (b4 + b5)$$

Інші параметри залишимо без змін.

Результати роботи програми:

```
73   === s27 with d9 d10 c7 right c8 left ===
74
75   Scheme type = greedy
76   time = 77
77   path = s27-d9-d10-c7-right-c8-left-greedy
78   sp = 0.9997598842949585, sq = 0.00024011570500589362
79   state count = 134217728
80
81   Scheme type = brute
82   time = 220
83   path = s27-d9-d10-c7-right-c8-left-brute
84   sp = 0.9997598842949585, sq = 0.00024011570500589362
85   state count = 134217728
```

Метрики:

```
In [16]: process_sample_with_del("s27-d9-d10-c7-right-c8-left-greedy")
```

```
=== sum probabilities ===
p = 0.9997598842949936
q = 0.00024011570500590327
p + q = 0.9999999999999996
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b2: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b4: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b5: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
c1: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c2: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c4: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c5: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
c6: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
d1: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d2: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d3: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d6: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d8: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
m1: sv1 = 4.31782520e-07, sv2 = 2.15963907e-07
m2: sv1 = 4.31782520e-07, sv2 = 2.15963907e-07
d9: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
d10: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
c7: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
c8: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
```

```
In [17]: process_sample_with_del("s27-d9-d10-c7-right-c8-left-brute")
```

```
=== sum probabilities ===
p = 0.9997598842949936
q = 0.00024011570500590327
p + q = 0.9999999999999996
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
a2: sv1 = 1.20000000e-04, sv2 = 1.20000000e-04
b1: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b2: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b4: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
b5: sv1 = 1.28222227e-08, sv2 = 3.82667817e-09
c1: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c2: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c4: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
c5: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
c6: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
d1: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d2: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d3: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d6: sv1 = 1.84762087e-08, sv2 = 5.28254551e-09
d8: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
m1: sv1 = 4.31782520e-07, sv2 = 2.15963907e-07
m2: sv1 = 4.31782520e-07, sv2 = 2.15963907e-07
d9: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
d10: sv1 = 1.84762194e-08, sv2 = 5.28255616e-09
c7: sv1 = 3.44398180e-07, sv2 = 9.85163153e-08
c8: sv1 = 3.44329373e-07, sv2 = 9.84474673e-08
```

1. Надійність незначно підвищилась: 0.99975988 проти 0.99975971
2. Жадібний алгоритм очікувано швидший: 77 секунд проти 220

3.7 Модифікація 6: 29 елементів (a3, a4)

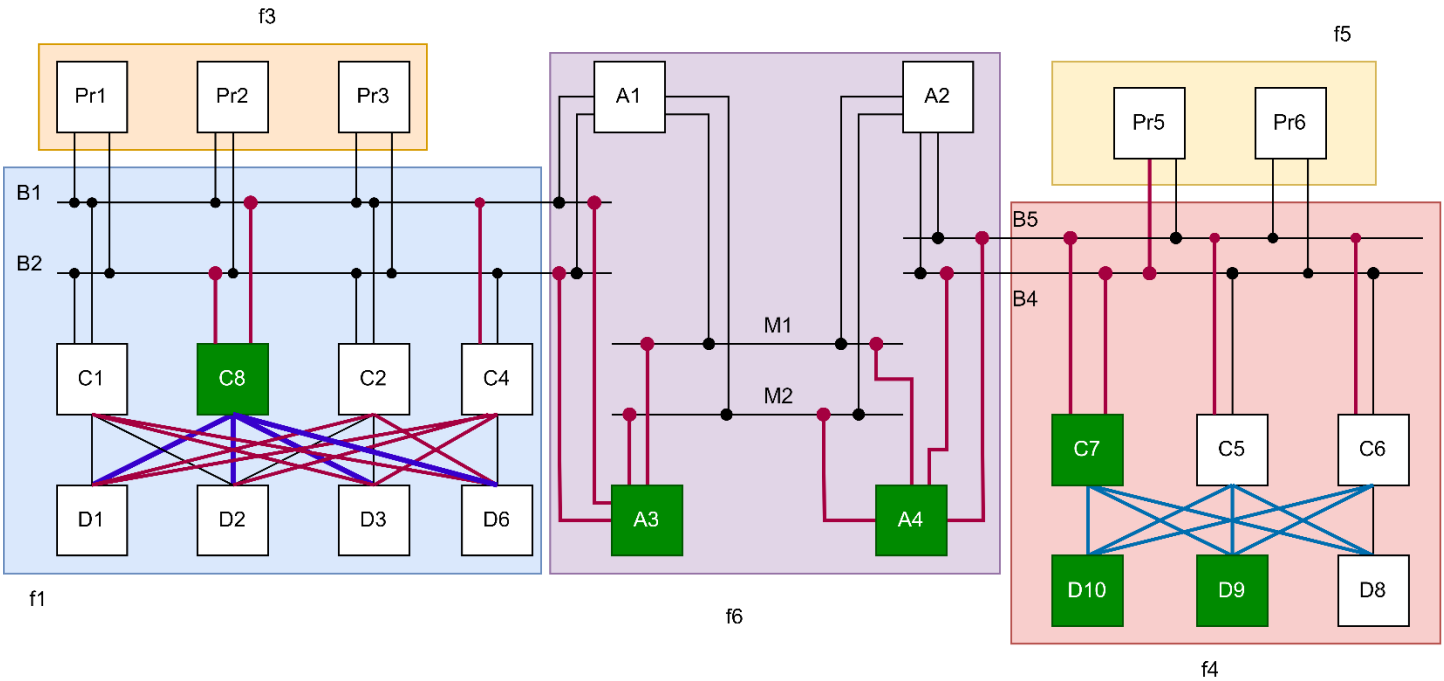


Рис. 11 Схема з елементами a3, a4 (всього 29 елементів)

Жодна із змін не впливала на контролери типу a. Саме контролери типу a, згідно метрик, є головною причиною відмови всієї схеми. Додамо 2 контролери: a3 та a4. Тепер права та ліва частини схеми мають декілька з'єднань з магістралями m1, m2.

Логічна функція прийме вид:

$$f1 = (d1 + d2 + d3 + d4) * (c1 + c8 + c2 + c4) * (b1 + b2)$$

$$f3 = pr1 * pr2 * pr3$$

$$f4 = (d10 + d9 + d8) * (c7 + c5 + c6) * (b4 + b5)$$

$$f5 = pr5 * pr6$$

$$f6 = (a1 + a3) * (a2 + a4) * (m1 + m2)$$

Інші параметри залишило без змін.

Результати роботи програми:

```
=== s29 with d9 d10 c7 right c8 left a4 ===

Scheme type = greedy
time = 397
path = s29-d9-d10-c7-right-c8-left-a4-greedy
sp = 0.999999841063663, sq = 1.5893623277066473e-07
state count = 536870912

Scheme type = brute
time = 1022
path = s29-d9-d10-c7-right-c8-left-a4-brute
sp = 0.999999841063663, sq = 1.5893623277066473e-07
state count = 536870912
```

Метрики:

```
In [18]: process_sample_with_del("s29-d9-d10-c7-right-c8-left-a4-greedy")

=== sum probabilities ===
p = 0.9999998410637558
q = 1.5893623277071066e-07
p + q = 0.9999999999999886
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
a2: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
b1: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b2: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b4: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b5: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
c1: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c2: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c4: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c5: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
c6: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
d1: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d2: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d3: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d6: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d8: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
m1: sv1 = 3.45480970e-07, sv2 = 1.29610557e-07
m2: sv1 = 3.45480970e-07, sv2 = 1.29610557e-07
d9: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
d10: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
c7: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
c8: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
a3: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
a4: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
```

```
In [19]: process_sample_with_del("s29-d9-d10-c7-right-c8-left-a4-brute")

=== sum probabilities ===
p = 0.9999998410637558
q = 1.5893623277071066e-07
p + q = 0.9999999999999886
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
a2: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
b1: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b2: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b4: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
b5: sv1 = 9.22508424e-09, sv2 = 2.27380633e-10
c1: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c2: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c4: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
c5: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
c6: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
d1: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d2: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d3: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d6: sv1 = 1.32003265e-08, sv2 = 3.49659736e-12
d8: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
m1: sv1 = 3.45480970e-07, sv2 = 1.29610557e-07
m2: sv1 = 3.45480970e-07, sv2 = 1.29610557e-07
d9: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
d10: sv1 = 1.32003371e-08, sv2 = 3.50724489e-12
c7: sv1 = 2.46074936e-07, sv2 = 1.34056587e-10
c8: sv1 = 2.46006113e-07, sv2 = 6.51921015e-11
a3: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
a4: sv1 = 8.63914139e-08, sv2 = 1.44173423e-08
```

1. Спостерігаємо значний приріст надійності: 0.9999998 проти 0.99975. Початкова мета: досягнути надійності, більшої або рівної 0.9999 – досягнута.
2. Імовірність відмов контролерів типу a зменшилась з четвертого до восьмого порядку.
3. Вкотре жадібний алгоритм знаходить варіанти реконфігурацій, які дозволяють досягати кінцевою надійності, ідентичної до повного перебору, за значно менший час.

4. Додаткове тестування швидкодії

Обчислення розділу 3 проводились на EC2 vm. Однією з помилок налаштування vm було використання мережевих дисків (EBS volumes). Через це результати порівняння швидкодії було спотворено потенційним впливом продуктивності мереж.

Незначний приріст надійності для модифікацій 4, 5 на кінцеву надійність системи, та суттєвий приріст надійності внаслідок додавання контролерів a3, a4, наводить на думку, що можна отримати надійність, схожу з модифікацією 6, не застосовуючи модифікації 4, 5:

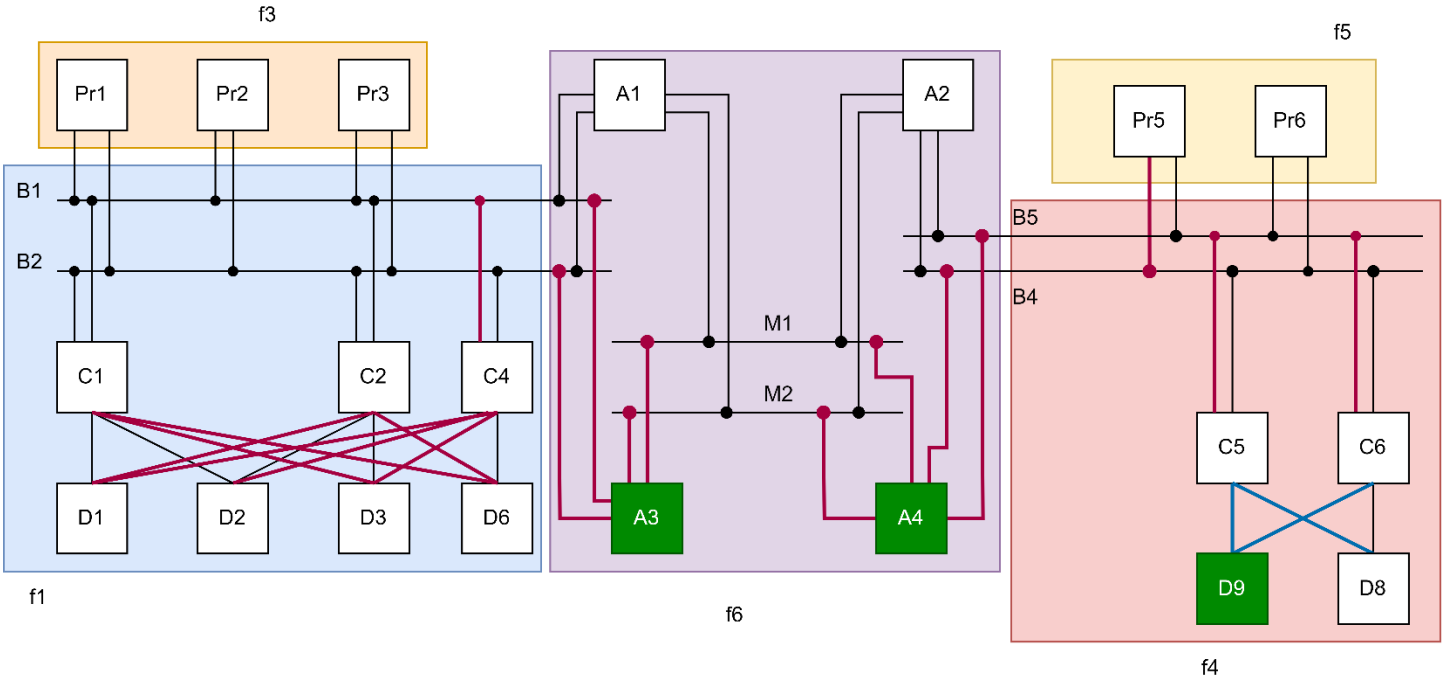


Рис.12 Модифікація 7 (26 елементів)

Логічна функція схеми:

$$f1 = (d1 + d2 + d3 + d4) * (c1 + c2 + c4) * (b1 + b2)$$

$$f3 = pr1 * pr2 * pr3$$

$$f4 = (d9 + d8) * (c5 + c6) * (b4 + b5)$$

$$f5 = pr5 * pr6$$

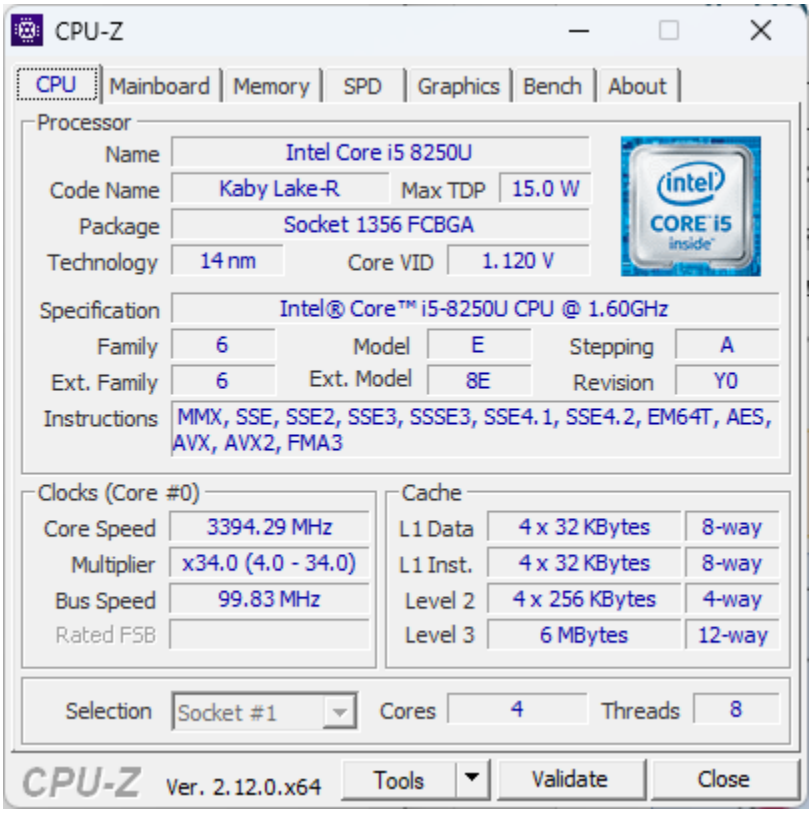
$$f6 = (a1 + a3) * (a2 + a4) * (m1 + m2)$$

Таблиця реконфігурацій є тією ж що і для модифікацій 1 – 6:

	Ln	Lm	Перерозподіли навантаження
pr1	50	100	[pr2 = 50], [pr3 = 50], [pr2 = 25, pr3 = 25], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr2	50	100	[pr1 = 50], [pr3 = 50], [pr1 = 25, pr3 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr3	50	100	[pr1 = 50], [pr2 = 50], [pr1 = 25, pr2 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30]

pr5	30	60	[pr6 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]
pr6	30	60	[pr5 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]

Для додаткового тестування швидкодії виконаємо обчислення для оригінальної схеми, модифікацій 1, 2, 3, 7. Обчислення здійснюватимемо на процесорі:



Результати виконання програми:

```
PS C:\Users\Bohdan\Programming\labs-5-1\SchemeReliability\x64\Release> .\sr-research.exe

=== original s23 ===

Scheme type = greedy
time = 4
path = s23-original-greedy
sp = 0.9990208569798784, sq = 0.0009791430201189963
state count = 8388608

Scheme type = brute
time = 3
path = s23-original-brute
sp = 0.9990208569798784, sq = 0.0009791430201189963
state count = 8388608

=== s23 with rt (7 7 7 8 8) ===

Scheme type = greedy
time = 9
path = s23-77788-greedy
sp = 0.9992607082975018, sq = 0.0007392917024915111
state count = 8388608

Scheme type = brute
time = 157
path = s23-77788-brute
sp = 0.9992607082975018, sq = 0.0007392917024915111
state count = 8388608

=== s23 with rt and modified connections ===

Scheme type = greedy
time = 10
path = s23-77788-modified-connections-greedy
sp = 0.9997377215216205, sq = 0.00026227847836073244
state count = 8388608

Scheme type = brute
time = 182
path = s23-77788-modified-connections-brute
sp = 0.9997377215216205, sq = 0.00026227847836073244
state count = 8388608

=== s24 with d9 right ===

Scheme type = greedy
time = 21
path = s24-d9-right-greedy
sp = 0.9997597157514958, sq = 0.00024028424848725245
state count = 16777216

Scheme type = brute
time = 308
path = s24-d9-right-brute
sp = 0.9997597157514958, sq = 0.00024028424848725245
state count = 16777216

=== s26 final ===

Scheme type = greedy
time = 106
path = s26-final-greedy
sp = 0.9999996724797207, sq = 3.2752016698960673e-07
state count = 67108864

Scheme type = brute
time = 1261
path = s26-final-brute
sp = 0.9999996724797207, sq = 3.2752016698960673e-07
state count = 67108864
PS C:\Users\Bohdan\Programming\labs-5-1\SchemeReliability\x64\Release>
```

Додатково обчислимо метрики для модифікації 7:

```
process_sample_with_del("s26-final-greedy")
[5] ✓ 1m 42.6s
```

```
=== sum probabilities ===
p = 0.9999996724798291
q = 3.2752016698965956e-07
p + q = 0.9999999999999961
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr2: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr3: sv1 = 1.20000000e-04, sv2 = 1.38217191e-11
pr4: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
pr5: sv1 = 1.20000000e-04, sv2 = 1.55488898e-11
a1: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
a2: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
b1: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b2: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b4: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b5: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
c1: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c2: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c4: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c5: sv1 = 4.14005421e-07, sv2 = 1.68165335e-07
c6: sv1 = 4.14005421e-07, sv2 = 1.68165335e-07
d1: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d2: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d3: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d6: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d8: sv1 = 1.36877320e-08, sv2 = 4.91194637e-10
m1: sv1 = 3.45541602e-07, sv2 = 1.29671226e-07
m2: sv1 = 3.45541602e-07, sv2 = 1.29671226e-07
d9: sv1 = 1.36877320e-08, sv2 = 4.91194637e-10
a3: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
a4: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
```

```
process_sample_with_del("s26-final-brute")
[6] ✓ 1m 30.8s
```

```
=== sum probabilities ===
p = 0.9999996724798291
q = 3.2752016698965956e-07
p + q = 0.9999999999999961
=== reconfiguration is consistent ===
=== sum fail probability(sv1 | sv2) per element ===
pr1: sv1 = 1.20000000e-04, sv2 = 1.72800000e-12
pr2: sv1 = 1.20000000e-04, sv2 = 8.63875589e-12
pr3: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
pr4: sv1 = 1.20000000e-04, sv2 = 6.22030234e-16
pr5: sv1 = 1.20000000e-04, sv2 = 5.18337794e-12
a1: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
a2: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
b1: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b2: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b4: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
b5: sv1 = 9.22761144e-09, sv2 = 2.29909354e-10
c1: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c2: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c4: sv1 = 2.46144014e-07, sv2 = 2.03175988e-10
c5: sv1 = 4.14005421e-07, sv2 = 1.68165335e-07
c6: sv1 = 4.14005421e-07, sv2 = 1.68165335e-07
d1: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d2: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d3: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d6: sv1 = 1.32040331e-08, sv2 = 7.20544391e-12
d8: sv1 = 1.36877320e-08, sv2 = 4.91194637e-10
m1: sv1 = 3.45541602e-07, sv2 = 1.29671226e-07
m2: sv1 = 3.45541602e-07, sv2 = 1.29671226e-07
d9: sv1 = 1.36877320e-08, sv2 = 4.91194637e-10
a3: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
a4: sv1 = 8.64116295e-08, sv2 = 1.44375699e-08
```

1. Різниця у швидкодії жадібного алгоритму та повного обходу стає ще більш очевидною.
2. Надійність модифікації 7 майже не поступається надійності модифікації 6. При цьому схема 7 має на 3 елементи менше ніж схема 6.

5. Ручні обчислення деяких BCC для початкової схеми та модифікації 1

Для додаткової перевірки правильності роботи програми порівняємо її результати з деякими порахованими вручну BCC. Отримання результатів для конкретних BCC із згенерованих .ssv файлів описано у файлі research.ipynb.

Метод виконання ручних розрахунків

На основі логічної функції системи був виведений список правил для спрощення розрахунків. Виконання або не виконання цих правил буде свідчити про стан системи $F(V_2)$.

Як виводилися правила

Спочатку розглянемо можливі BCC з кратністю відмов 1.

Випишемо повністю логічну функцію та виділимо критично важливі для роботи системи елементи (такі, при відмові хоча б одного з яких система гарантовано стане неробочою):

F = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) *d6 * c4 * b2 * pr1 * pr2* pr3 * d8 * (c5 + c6) * b4 * pr5 * pr6 * b5 * a1 * a2 * (m1 + m2)

Як можна побачити, критично важливими елементами є ті елементи, для яких не забезпечена пасивна відмовостійкість, тобто паралельне з'єднання. Процесори були виділені жовтим кольором, бо для них існує таблиця перерозподілів навантаження, тому навіть при відмові одного з них система $F(V_2)$ все ще може виявитися робочою.

Таким чином одразу можна визначити, що будь-який BCC, у якого хоча б один критично важливий елемент (червоний) неробочий, в результаті матиме $F(V_2) = 0$.

Розглянемо кілька можливих варіантів векторів із кратністю відмов 1, визначимо стан системи $F(V_2)$ за класичним алгоритмом та перевіримо правильність вищенаведених суджень.

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f4 = d8 * (c5 + c6) * b4 = 0 * (1 + 1) * 1 = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f2 = d6 * c4 * b2 = 1 * 0 * 1 = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
F(V2)	f6 = a1 * a2 * (m1 + m2) = 0 * 1 * (1 + 1) = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
F(V2)	f2 = d6 * c4 * b2 = 1 * 1 * 0 = 0																						

Рис. 13 Приклади неробочих ВСС з кратністю відмов 1

Значення ВСС згідно програми:

```
vector 1 type 1 = False
vector 2 type 1 = False
vector 3 type 1 = False
vector 4 type 1 = False
```

У кожному випадку на рисунку 1 через несправність одного “червоного” елемента не працювала вся схема. Це можна довести без розрахунку всього стану системи F, адже при відмові хоча б однієї функції fi система стає неробочою (через кон’юнкцію). Також варто зазначити, що в усіх зображених випадках перерозподіл навантаження не відбувається, адже він застосовується лише для процесорів, що буде розглянуто пізніше.

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((0 + 1) * 1 + (1 + 1) * 1) * (1 + 1) = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f4 = d8 * (c5 + c6) * b4 = 1 * (0 + 1) * 1 = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 1) * 1 + (1 + 1) * 1) * (0 + 1) = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
F(V2)	f6 = a1 * a2 * (m1 + m2) = 1 * 1 * (1 + 0) = 1																						

Рис. 14 Приклади роботоздатних ВСС з кратністю 1

Значення ВСС згідно програми:

```
vector 5 type 1 = True
vector 6 type 1 = True
vector 7 type 1 = True
vector 8 type 1 = True
```

На рисунку 14 видно, що при неробочому стані “зеленого” елемента стан системи $F(V_2) = 1$, тобто вона працездатна.

Перейдемо до ВСС з кратністю відмов 2.

На цьому етапі ми будемо розглядати лише елементи, що мають пасивну відмовостійкість, адже, як було з’ясовано раніше, наявність хоча б одного критично важливого елемента (“червоного”) у парі непрацюючих елементів гарантує відмову всієї системи.

Прибравши критичні елементи, отримуємо функцію F_1:

F_1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) * (c5 + c6) * (m1 + m2)

Розглянемо кожне паралельне з'єднання і зробимо певні висновки:

- Пара d1 і d2 паралельно під'єднана до c1, а пара d2 і d3 – до c2. Навіть при відмові обох елементів у одній з пар, система буде працездатною, бо у такому випадку спрацює друга пара, і навпаки. Це досягається завдяки диз'юнкції всередині пари та між ними. В результаті маємо, що при відмові двох з трьох елементів D1, D2 та D3 стан системи $F(V_2)$ буде робочим.
- Контролери c1 і c2 під'єднані паралельно до двох шин і до набору датчиків з одним спільним елементом. Вони виконують аналогічну роль у функції f1, що можна побачити по диз'юнкції між ними. Згідно з даною функцією, якщо обидва контролери відмовлять, то $F(V_2)$ буде дорівнювати 0. Це також видно зі схеми, адже іншого варіанту доставки даних з датчиків на шину не існує.
- Контролери c5 та c6 під'єднані паралельно до одного датчика d8 і до шини b4. Згідно функції f4 та малюнку схеми при c5 = 0 і при c6 = 0 стан системи $F(V_2)$ буде неробочим.
- Магістралі m1 та m2 під'єднані паралельно до адаптерів a1 й a2. Згідно функції f6 та малюнку схеми при m1 = 0 і m2 = 0 стан системи $F(V_2)$ буде неробочим.
- Шини b1 і b2 під'єднані паралельно до трьох процесорів, двох контролерів та до адаптера. Проте до шини b2 додатково під'єднаний контролер c4, що робить ці шини не тотожними. Вище ми визначили шину b2 як критичний елемент системи саме через цей фактор: справність b2 є обов'язковою умовою для роботи функції f2. У цьому розділі вона з'явилася лише через те, що є ідентичною до шини b1 з точки зору з'єднань до елементів функції f1. Отже, дані шини не є паралельно під'єднаними, b2 є критично важливим (”червоним”) елементом, а відмова b1 не призводить до неробочого стану системи, бо її функції візьме на себе шина b2. Оберненою ситуації не може бути через те що b1 не з'єднана з c4.

Отже, повний список умов (правил) для кратності відмов 2:

1. Якщо хоча б один елемент, що відмовив, є критично важливим (“червоним”), система F(V2) не буде працювати;
2. Якщо елементи, що відмовили, складають одну з наступних пар: c1 і c2, c5 і c6, m1 і m2, то система F(V2) не буде працювати.
3. В інших випадках F(V2) буде працювати.

Перевіримо вищезазначене через підрахунки класичним алгоритмом.

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 1) * 0 + (1 + 1) * 0) * (1 + 1) = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
F(V2)	f6 = a1 * a2 * (m1 + m2) = 1 * 1 * (0 + 0) = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
V2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((0 + 1) * 1 + (1 + 1) * 1) * (1 + 0) = 1 && f2 = d6 * c4 * b2 = 1 * 1 * 0 = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f2 = d6 * c4 * b2 = 0 * 1 * 1 = 0 && f4 = d8 * (c5 + c6) * b4 = 1 * (0 + 1) * 1 = 1																						

Рис. 15 Приклади неробочих ВСС із кратністю 2

Значення ВСС згідно програми:

```
vector 1 type 2 = False
vector 2 type 2 = False
vector 3 type 2 = False
vector 4 type 2 = False
```

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 0) * 1 + (0 + 0) * 1) * (1 + 1) = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
V2	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 1) * 1 + (1 + 1) * 1) * (0 + 1) = 1 && f4 = d8 * (c5 + c6) * b4 = 1 * (1 + 0) * 1 = 1																						

Рис. 16 Приклади роботоздатних ВСС із кратністю 2

Значення ВСС згідно програми:

```
vector 5 type 2 = True
vector 5 type 2 = True
```

На рисунку 15 видно, що F(V2) = 0, якщо елементи, що відмовили, складають одну з перелічених вище пар. Також була повторно перевірена і підтверджена перша умова: при наявності хоча б одного несправного “червоного” елементу стан системи F(V2) = 0.

На рисунку 16 зображені приклади роботоздатних векторів станів системи: у першому випадку відмовили 2 з 3 датчиків, а у другому відмовила шина b1 і контролер c6. Як можна побачити, в обох випадках система робоча, а отже відмова лише одного елемента з “фіолетової” пари не призводить до відмови всієї системи.

Перейдемо до ВСС з кратністю відмов 3.

Проаналізуємо всі функції f_i. У всіх функціях крім f_1 наявні 3 або 4 елементи, які є або критичними елементами, або парою з паралельним з'єднанням (фіолетовий колір на рисунках). Це означає, що в кожній з цих окремо взятих функцій 2 відмови будь-яких елементів будуть означати відмову всієї системи (у функціях f_3 та f_5 є ситуації, за яких система буде робочою через перерозподіл навантаження процесорів, але при трьох відмовах система точно буде непрацездатною). Тобто у цих функціях точно буде виконуватися одна з вже наявних умов (правил), а значить їх подальший розгляд немає сенсу. Функція f_1 натомість має аж 7 елементів і досить гнучку структуру, і відмова одразу трьох елементів у межах цієї функції створює нові ситуації, які варто описати.

Можливі 2 ситуації з трьома відмовами в функції f_1:

1) Відмова всіх датчиків d1, d2, d3;

2) Відмова двох датчиків з одного боку паралельного з'єднання і контролеру з іншого: d1 = d2 = c2 = 0 або d2 = d3 = c1 = 0.

Отже, повний набір правил для визначення працездатності ВСС для кратності відмов 3 виглядає так:

1. Якщо хоча б один елемент, що відмовив, є критично важливим ("червоним"), система F(V2) не буде працювати;
2. Якщо серед трьох елементів, що відмовили, наявна одна з наступних пар: c1 і c2 або c5 і c6 або m1 і m2, то система F(V2) не буде працювати.
3. Якщо трьома елементами, що відмовили, є: d1, d2, d3 або d1, d2, c2 або d2, d3, c1, то система F(V2) не буде працювати.
4. В інших випадках F(V2) буде працювати.

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((0 + 0) * 1 + (0 + 0) * 1) * (1 + 1) = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((0 + 0) * 1 + (0 + 1) * 0) * (1 + 1) = 0																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 0) * 0 + (0 + 0) * 1) * (1 + 1) = 0																						

Рис. 17 Приклади неробочих ВСС із кратністю 3

Значення ВСС згідно програми:

```
vector 1 type 3 = False
vector 2 type 3 = False
vector 3 type 3 = False
```

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((0 + 0) * 0 + (0 + 1) * 1) * (1 + 1) = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	0	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
V2	1	0	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 0) * 0 + (0 + 1) * 1) * (1 + 1) = 1																						

	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
V2	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
F(V2)	f1 = ((d1 + d2) * c1 + (d2 + d3) * c2) * (b1 + b2) = ((1 + 1) * 0 + (1 + 1) * 1) * (1 + 1) = 1																						

Рис. 18 Приклади робочих ВСС з кратністю відмов 3

Значення ВСС згідно програми:

```
vector 4 type 3 = True
vector 5 type 3 = True
vector 6 type 3 = True
```

При розгляді кратності відмов 4 і більше нових правил не виникає через відносну простоту і невелику кількість елементів у схемі. Це означає, що при кратності відмов 4 гарантовано виникне одне з вищеописаних правил і F(V2) буде дорівнювати 0 або не виникне, і $F(V_2)$ буде дорівнювати 1.

Тепер розглянемо можливі ситуації при відмові процесорів.

Перерозподіл навантаження

Розглянемо можливі ситуації для початкової схеми і таблиці перерозподілів навантаження та для модифікації 1 (та сама схема та покращена таблиця перерозподілів навантаження).

Початкова схема

Таблиця реконфігурації:

	Ln	Lm	Перерозподіли навантаження
pr1	50	80	[pr2 = 25, pr3 = 25]
pr2	50	80	[pr1 = 25, pr3 = 25]
pr3	50	80	[pr1 = 25, pr2 = 25]
pr5	30	60	відсутні
pr6	30	60	відсутні

Для початкової схеми перерозподіл навантаження доступний лише для процесорів pr1, pr2, pr3 та лише при кратності відмов ВСС 1. Інші ситуації неможливі відповідно до таблиці.

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
Δpr											+25	-	+25	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
Δpr											-	-	-	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
F(V2)	f5 = pr5 * pr6 * b5 = 1 * 0 * 1 = 0																						

Рис. 19 Можливі варіанти перерозподілу навантаження для початкової схеми і таблиці реконфігурації

Значення ВСС згідно програми:

```
v2 compare (original reconfiguration)
vector 1 = True
vector 2 = False
```

Модифікація 1

Модифікована таблиця реконфігурації:

	Ln	Lm	Перерозподіли навантаження
pr1	50	100	[pr2 = 50], [pr3 = 50], [pr2 = 25, pr3 = 25], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr2	50	100	[pr1 = 50], [pr3 = 50], [pr1 = 25, pr3 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr3 = 25, pr5 = 30], [pr3 = 25, pr6 = 30]
pr3	50	100	[pr1 = 50], [pr2 = 50], [pr1 = 25, pr2 = 25], [pr1 = 25, pr5 = 30], [pr1 = 25, pr6 = 30], [pr2 = 25, pr5 = 30], [pr2 = 25, pr6 = 30]
pr5	30	60	[pr6 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]
pr6	30	60	[pr5 = 30], [pr1 = 35], [pr2 = 35], [pr3 = 35], [pr1 = 18, pr2 = 18], [pr2 = 18, pr3 = 18], [pr1 = 18, pr3 = 18], [pr1 = 12, pr2 = 12, pr3 = 12]

Для модифікації 1 розглянемо кратність відмов ВСС серед процесорів до чотирьох.

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
Δpr											+50	-	-	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
Δpr											-	-	-	-	+30								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f5 = pr5 * pr6 * b5 = 1 * 1 * 1 = 1																						

Рис. 20 Можливі варіанти перерозподілу навантаження ВСС з кратністю відмов 1

Значення ВСС відповідно програми:

```
vector 1 type 1 = True
vector 2 type 1 = True
```

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
Δpr											-	-	+50	+30	+30								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1
Δpr											+35	+35	-	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f5 = pr5 * pr6 * b5 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1
Δpr											-	+50	-	-	+30								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 1 * 1 = 1 && f5 = pr5 * pr6 * b5 = 1 * 1 * 1 = 1																						

Рис. 21 Можливі варіанти перерозподілу навантаження ВСС з кратністю відмов 2

Значення ВСС відповідно програми:

```
vector 1 type 2 = True
vector 2 type 2 = True
vector 3 type 2 = True
```

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1
Δpr											-	-	-	-	-								
V2	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 0 * 0 * 0 = 0																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1	1	1
Δpr											-	-	+50	+30	-								
V2	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 0 * 1 = 0 && f5 = pr5 * pr6 * b5 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1	1	1	1	1
Δpr											-	+50	+35	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 1 * 1 = 1 && f5 = pr5 * pr6 * b5 = 1 * 0 * 1 = 0																						

Рис. 22 – Можливі варіанти перерозподілу навантаження ВСС з кратністю відмов 3

Значення ВСС згідно програми:

```
vector 1 type 3 = False
vector 2 type 3 = False
vector 3 type 3 = False
```

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1
Δpr											-	-	-	-	+30								
V2	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 0 * 0 * 0 = 0 && f5 = pr5 * pr6 * b5 = 1 * 1 * 1 = 1																						

	Δpr - зміна навантаження на процесор після перерозподілу										Перерозподіл завжди			Перерозподіл у мод1								
	D1	D2	D3	D6	D8	C1	C2	C4	C5	C6	Pr1	Pr2	Pr3	Pr5	Pr6	A1	A2	B1	B2	B4	B5	M1	M2
V1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	1	1	1	1	1	1	1
Δpr											-	-	+50	-	-								
V2	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	1	1	1	1	1	1	1	1
F(V2)	f3 = pr1 * pr2 * pr3 = 1 * 0 * 1 = 0 && f5 = pr5 * pr6 * b5 = 0 * 0 * 1 = 0																						

Рис. 23 Можливі варіанти перерозподілу навантаження ВСС з кратністю відмов 4

Як можна побачити з рисунків 20-21, при відмові будь-яких двох процесорів при кратності відмов у ВСС 2 система буде гарантовано працездатною. У свою чергу рисунки 22-23 свідчать про те, що стан системи $F(V_2) = 0$, якщо відмовили 3 або більше процесори.

Висновки

В роботі було досліджено схему варіанту 17. Всі обчислення надійності проводились для повної множини векторів станів v1. Надійність початкової схеми приблизно рівна 0.99902085.

Для покращення надійності схеми було запропоновано модифікації:

- модифікація 1: просунута таблиця реконфігурацій, p ≈ 0.999260708
- модифікація 2: додаткові з'єднання елементів, p ≈ 0.99973772
- модифікація 3: додатковий елемент d9, p ≈ 0.9997597157515095
- модифікація 4: додатковий елемент d10, p ≈ 0.9997597162353798
- модифікація 5: додаткові елементи c7, c8, p ≈ 0.9997598842949585
- модифікація 6: додаткові елементи a3, a4, p ≈ 0.999999841063663
- модифікація 7: вилучення елементів c7, c8, d10, p ≈ 0.9999996724797207

Початкова мета: підвищити надійність схеми до значення, більшого або рівного 0.9999, досягнута.

Модифікації 4, 5 демонструють, що наївне додавання елементів до паралельних з'єднань не дозволяє підвищувати надійність нескінченно. Спостерігається деяка подібність залежності надійності від кількості елементів у паралельних з'єднаннях, із поведінкою збіжного числового ряду.

В роботі запропоновано 2 алгоритми реконфігурації: алгоритм повного перебору та жадібний алгоритм на основі оцінки навантаження. Час роботи програми з різними алгоритмами наведено в таблицях:

	greedy (EC2, секунди)	brute (EC2, секунди)
початкова схема	3	3
модифікація 1	2	18
модифікація 2	3	13
модифікація 3	1	35
модифікація 4	8	61
модифікація 5	77	220
модифікація 6	397	1022
модифікація 7	-	-

	greedy (local, секунди)	brute (local, секунди)
початкова схема	4	3
модифікація 1	9	157
модифікація 2	10	182
модифікація 3	21	308
модифікація 7	106	1261

Згідно досліджень, результати обчислень надійності із використанням жадібного алгоритму не відрізняються від значень надійності, отриманими із використання повного перебору. При цьому

жадібний алгоритм в середньому працює у 12.74 (local) разів швидше за алгоритм повного перебору.

Експериментально підтверджено властивість алгоритму повного перебору, у випадках, коли не існує реконфігурації, що утворить робочий v2 із даного неробочого v1, знаходити вектор v2 із найменшою кількістю неробочих елементів. Жадібний алгоритм не володіє такою властивістю. Ця відмінність алгоритмів призводить до дещо кращих значень метрик для повного перебору. Разом з тим, значення надійності схеми співпадають. З точки зору обчислення надійності схеми, якщо схема відмовила – кількість елементів, що відмовили, не має значення.