Rozmieszczanie procesów przetwarzania danych w systemach IoT

(zjawisko osmozy)

Krzysztof Jędrychowski
Paweł Kiełbasa
Tomasz Kłusak
Artur Pietrzyk

Etap I

Zapoznanie się ze specyfikacją

W przygotowaniu do wykonania projektu zapoznaliśmy się z załączonymi artykułami dotyczącymi modelowania systemów IoT oraz dotyczących wykorzystania koncepcji zjawiska osmozy w tym modelowaniu. Zapoznaliśmy się z podziałem systemu na poszczególne warstwy (Cloud, Edge/FOG, IoT Devices) i rolą urządzeń w poszczególnych warstwach

Zapoznanie się z symulatorem IotSim-Edge

Zgodnie z planem projektu, w celu wykonania symulacji pobraliśmy i uruchomiliśmy symulator IotSim-Edge. Jest to symulator który umożliwia testowanie scenariuszy edge computing. Za pomocą plików konfiguracyjnych json definiuje się urządzenia systemu i ich działanie. Sam symulator po wczytaniu konfiguracji wykonuje testy i prezentuje wyniki w postaci tekstowej w konsoli.

Symulator: https://github.com/DNJha/IoTSim-Edge

Dokumentacja:
https://github.com/DNJha/IoTSim-Edge/blob/master/IoTSim-Edge_User_Manual.pdf

Wnioski

Okazało się iż symulator nie sprawdzi się w naszym przypadku – jest skierowany do przeprowadzania innego rodzaju symulacji i nie ma zbyt wielu mechanizmów kontrolowania przetwarzania danych i load balancingu. Skupia się on na warstwach IoT Devices i Edge, nie umożliwia natomiast kontroli nad warstwą Cloud. Aby zamodelować rozmieszczenie procesów z wykorzystaniem zjawiska osmozy musimy znaleźć alternatywne narzędzia, lub stworzyć własne proste symulacje, prezentujące metodę osmozy.

Etap II

Podstawowe założenia

W tej części skupiliśmy się na stworzeniu algorytmu przekazywania stężeń dla listy urządzeń wraz z podanymi stężeniami oraz opóźnieniami.

Stworzenie projektu

Projekt wykonaliśmy w Java wraz z wykorzystaniem Gradla. Założyliśmy, że dane o urządzeniach będą przekazywane za pomocą pliku configuration.json w którym dla każdego urządzenia znajdują się informacje o nazwie urządzenia, początkowym stężeniu, opóźnieniu i liście sąsiadów. Na tym etapie potwierdziliśmy czy dane urządzenia są wykrywalne przez kod poprzez wypisanie ich specyfikacji.

```
"devices": [
    {
        "name": "dev1",
        "concentration": 0.5,
        "delay": 3,
        "neighbours": [ "dev2", "dev3" ]
    },
    {
        "name": "dev2",
        "concentration": 0.65,
        "delay": 5,
        "neighbours": [ "dev1", "dev3" ]
    },
    {
        "name": "dev3",
        "concentration": 0.1,
        "delay": 2.5,
        "neighbours": [ "dev1", "dev2", "dev4" ]
    },
    {
        "name": "dev4",
        "concentration": 0.3,
        "delay": 2,
        "neighbours": [ "dev3" ]
    }
]
```

Rysunek 1 Urządzenia w zastosowanym symulatorze

Efekt wywołania polecenia wypisującego wszystkie urządzenia w symulatorze został pokazany poniżej

```
Configuration{devices=[ConfigurationDevice{name='dev1', concentration=0.5, delay=3.0, neighbours=[dev2, dev3]}
, ConfigurationDevice{name='dev2', concentration=0.65, delay=5.0, neighbours=[dev1, dev3]}
, ConfigurationDevice{name='dev3', concentration=0.1, delay=2.5, neighbours=[dev1, dev2, dev4]}
, ConfigurationDevice{name='dev4', concentration=0.3, delay=2.0, neighbours=[dev3]}
]}
```

Rysunek 2 Wynik z konsoli o urządzeniach w symulatorze

Stworzenie symulatora wraz z obsługą opóźnień

W celu stworzenia symulacji urządzeń stworzyliśmy nowe klasy służące do obsługi urządzeń, wysyłania informacji oraz samej symulacji. Ponieważ dla każdego urządzenia musieliśmy dodać obsługę przenoszenia stężeń oraz wysyłania informacji postanowiliśmy stworzyć nową klasę reprezentującą urządzenia. Lista urządzeń zawarta jest w klasie Simulator która odpowiada za uruchomienie symulacji, obsługę opóźnień dla każdego urządzenia i wyświetlaniu stężeń dla każdego urządzenia w danej iteracji. W celu obsługi opóźnień wykorzystaliśmy funkcję scheduleAtFixedRate z klasy Timer. Implementacja opóźnień i wynik z konsoli poniżej

```
public void start() {
    simulationDevices.forEach(
        device -> {
            long timeout = (long)(device.getDelay() * 1000);
            TimerTask task = () -> {
                    device.handleTimeout();
                    printStatus();
            };
            timer.scheduleAtFixedRate(task, timeout, timeout);
        }
    );
}
```

```
Iteration 0:
Device: dev1 Concentration: 0.5
Device: dev2 Concentration: 0.65
Device: dev3 Concentration: 0.1
Device: dev4 Concentration: 0.3
```

Stworzenie algorytmu przekazywania stężeń

W naszym algorytmie urządzenie będzie przekazywało siebie do każdego z sąsiadów, który będzie porównywał ze sobą stężenia i doprowadzał do wyrównania stężeń, wypisując informację o zmianie. Wysyłanie wiadomości obsłużone zostało w klasie MessageSender

```
public void send(SimulationDevice from, String to) {
    SimulationDevice toDevice = deviceMap.get(to);
    toDevice.handleMessage(from);
}
```

Obsługę przekazywania stężeń oraz rozsyłania wiadomości obsłużyliśmy w klasie SimulationDevice

```
public void handleMessage(SimulationDevice device) {
    double relative = this.getConcentration() - device.getConcentration();
    if(relative<0) {
        device.setRelativeConcentration(relative * 0.5);
        this.setRelativeConcentration(-relative * 0.5);
        System.out.println("Change:\n\tFrom: "+device.getName()+"\n\tTo: "+ this.getName() +"\n\tValue: "+(-relative*0.5) + "\n\n");
    }
}
```

```
public void handleTimeout() {
    neighbours.forEach(
        neighbour -> {
            messageSender.send(this, neighbour);
        }
    );
}
```

Wyniki

Dla urządzeń wymienionych na Rys. 1 algorytm uzyskał porównywalne stężenia (równe do dwóch cyfr znaczących) po ok. 20 sekundach. Wyniki poniżej

```
Time from start 20.0048007 s:
Device: dev1 Concentration: 0.38980712890625
Device: dev2 Concentration: 0.3875
Device: dev3 Concentration: 0.3875
Device: dev4 Concentration: 0.38519287109375
```

Natomiast proces zakończenia przekazywania stężeń zakończył się po około 140 sekundach na poniższych wartościach

```
Time from start 139.9964858 s:
Device: dev1 Concentration: 0.38749999999999996
Device: dev2 Concentration: 0.38749999999999996
Device: dev3 Concentration: 0.38749999999999996
Device: dev4 Concentration: 0.38749999999999996
```

Wnioski

Algorytm doprowadza do poprawnego rozkładu stężeń. W miarę równomierne wyniki są osiągane dosyć szybko, natomiast ze względu na sposób zmiany wartości (doprowadzanie do równości między dwoma urządzeniami) czas do zrównania wartości stężeń jest długi w porównaniu do w miarę równych wyników.

Etap III

1. Cel

W trzecim etapie celem było stworzenie bardziej realistycznego modelu stężeń, dodanie graficznej reprezentacji urządzeń oraz przetestowanie różnych scenariuszy.

2. Realizacja

a. Zmiana modelu koncentracji

Zmodyfikowaliśmy klasę reprezentującą urządzenia, wprowadzając zamiast dotychczasowego stężenia dwie wielkości: ilość obecnie obsługiwanych zadań i pojemność urządzenia. Na podstawie tych wielkości obliczamy stężenie, które wykorzystujemy w algorytmie przekazywania stężeń.

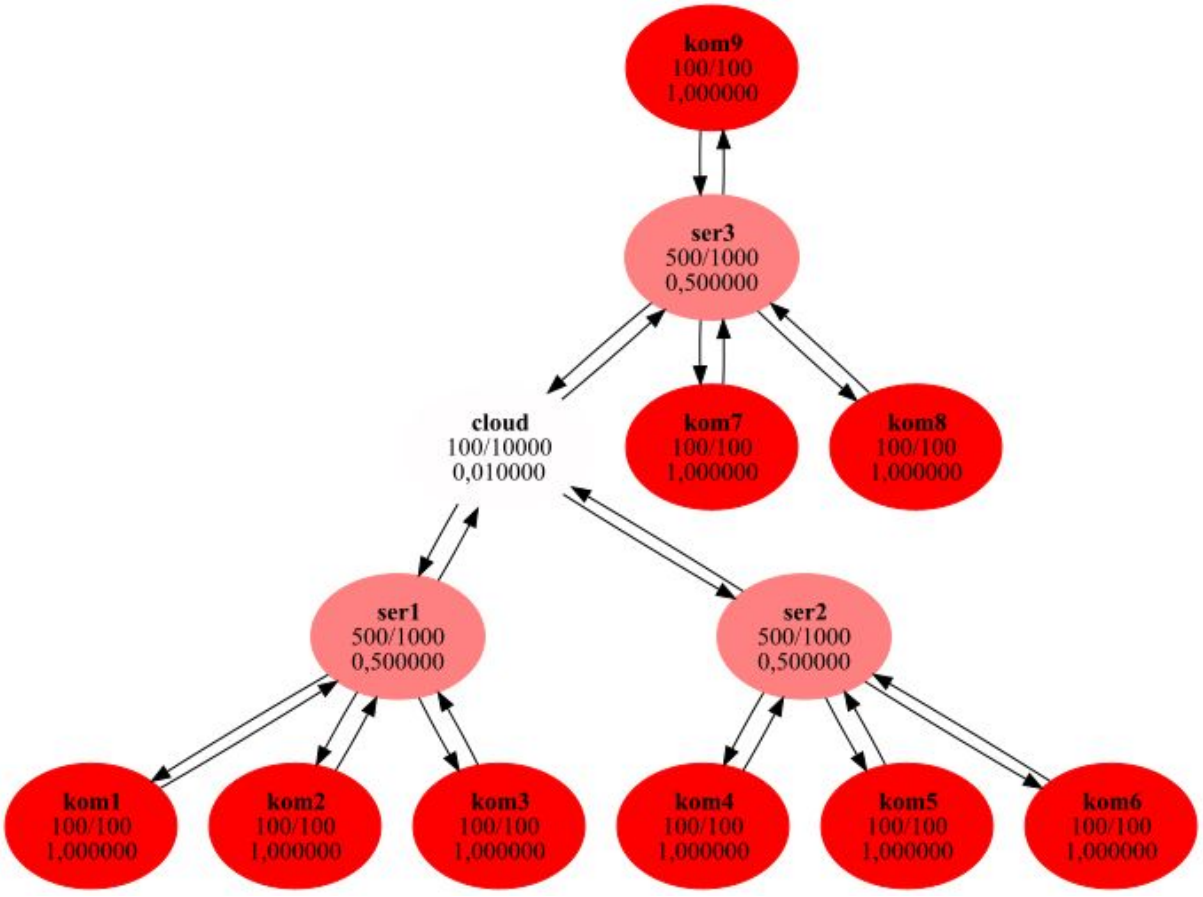
Zmodyfikowaliśmy również sposób wyświetlania informacji o urządzeniach - dodaliśmy odrębną klasę, StatusLogger, która odpowiada za wyświetlanie informacji o stanie urządzeń i komunikatów o przenoszeniu zadań między urządzeniami.

b. Graficzne przedstawianie urządzeń

Dodaliśmy klasę GraphBuilder odpowiedzialną za tworzenie grafu przedstawiającego połączenia między urządzeniami i najważniejsze informacje o każdym urządzeniu. Do sporządzania grafów wykorzystaliśmy oprogramowanie graphviz. Utworzone grafy pozwalają na obserwowanie zmian zachodzących w systemie. Dla udogodnienia wygenerowane grafy zapisujemy również w postaci pojedynczego gifa, z wykorzystaniem zewnętrznego kodu.

W grafie wyświetlane są: nazwa urządzenia, ilość obecnych zadań i pojemność urządzenia, obecne stężenie urządzenia, oraz połączenia między urządzeniami.

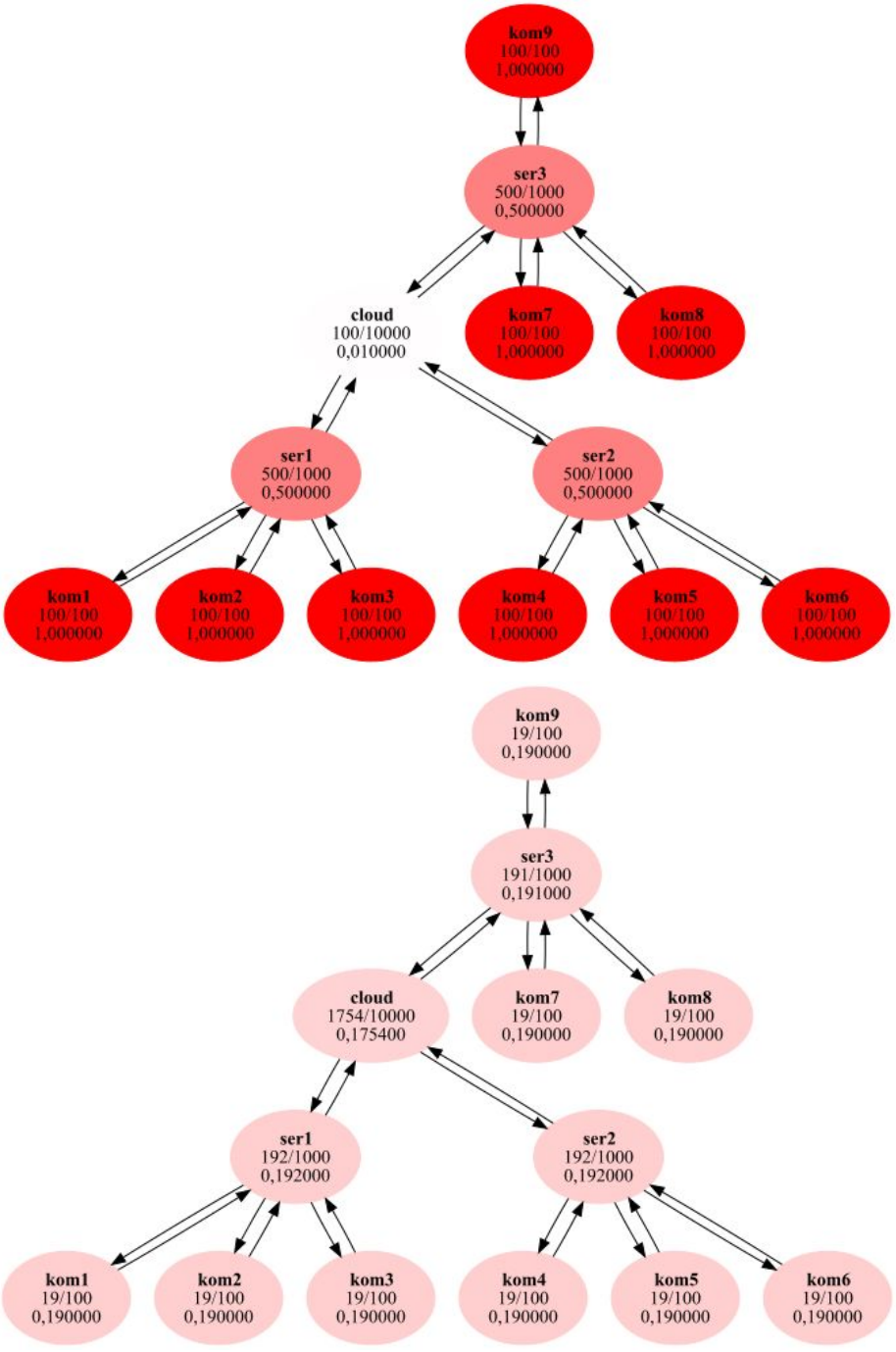
Przykład grafu:



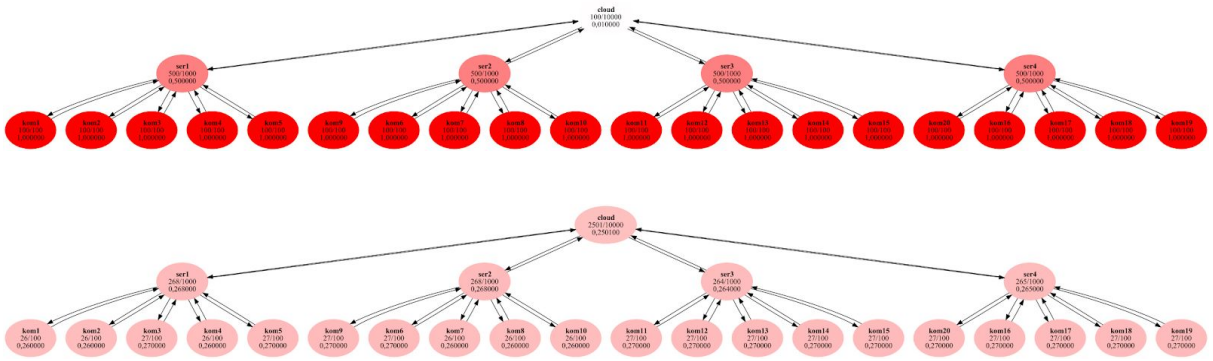
c. Testowanie różnych scenariuszy

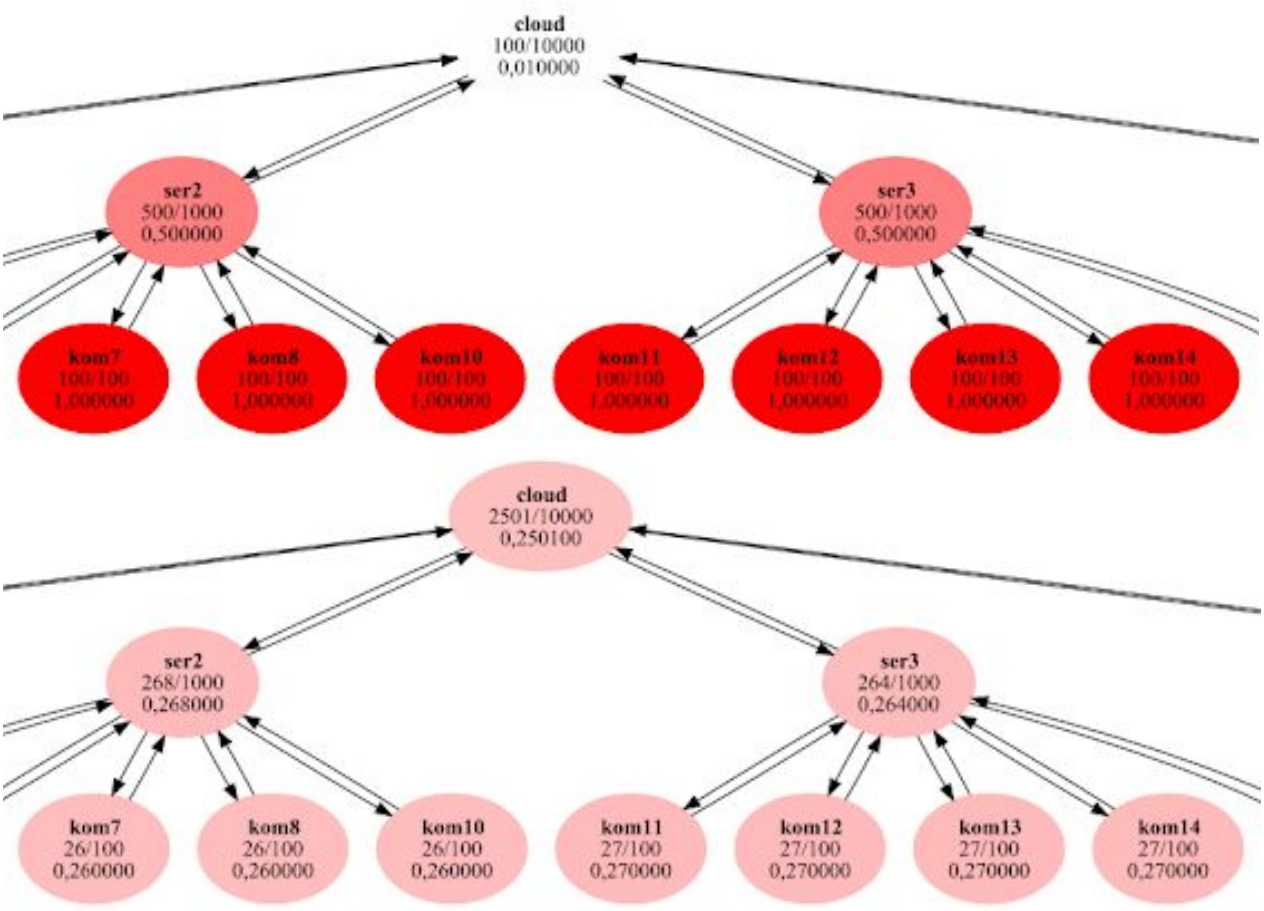
Utworzyliśmy cztery scenariusze o różnych konfiguracjach. Przyjęliśmy, że wymiana zadań między urządzeniami zachodzi co 2s.

- Scenariusz 1 (równowaga uzyskana po upływie 20134s):

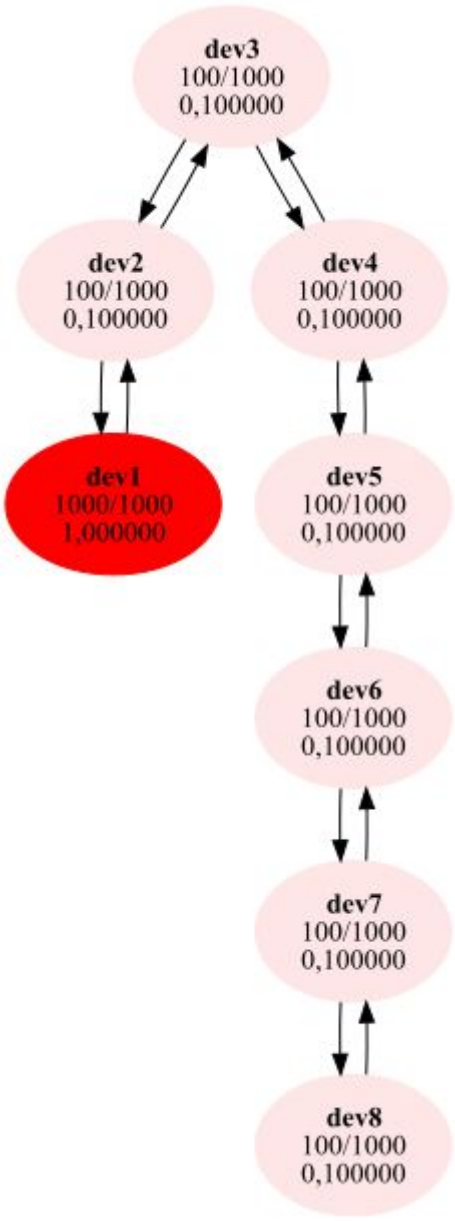


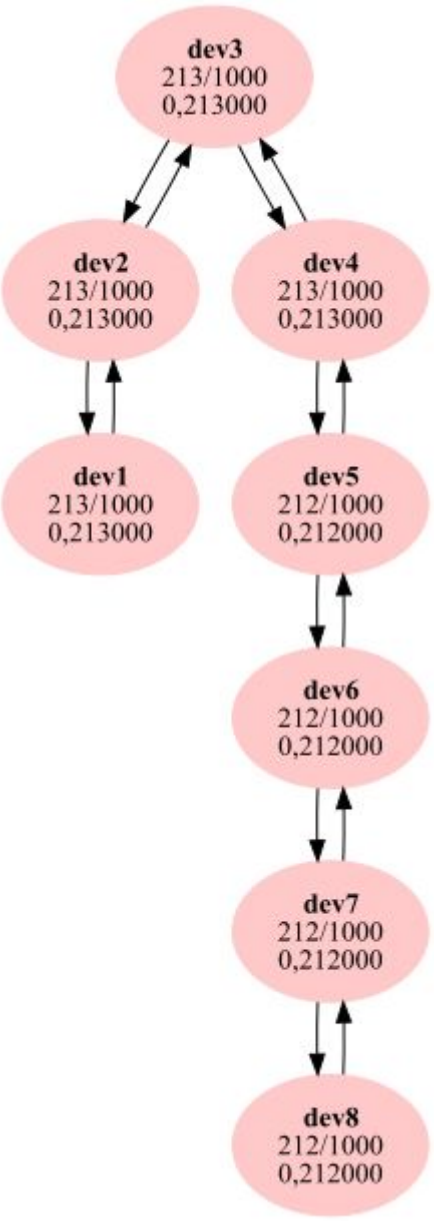
- Scenariusz 2 (równowaga po 48350s)



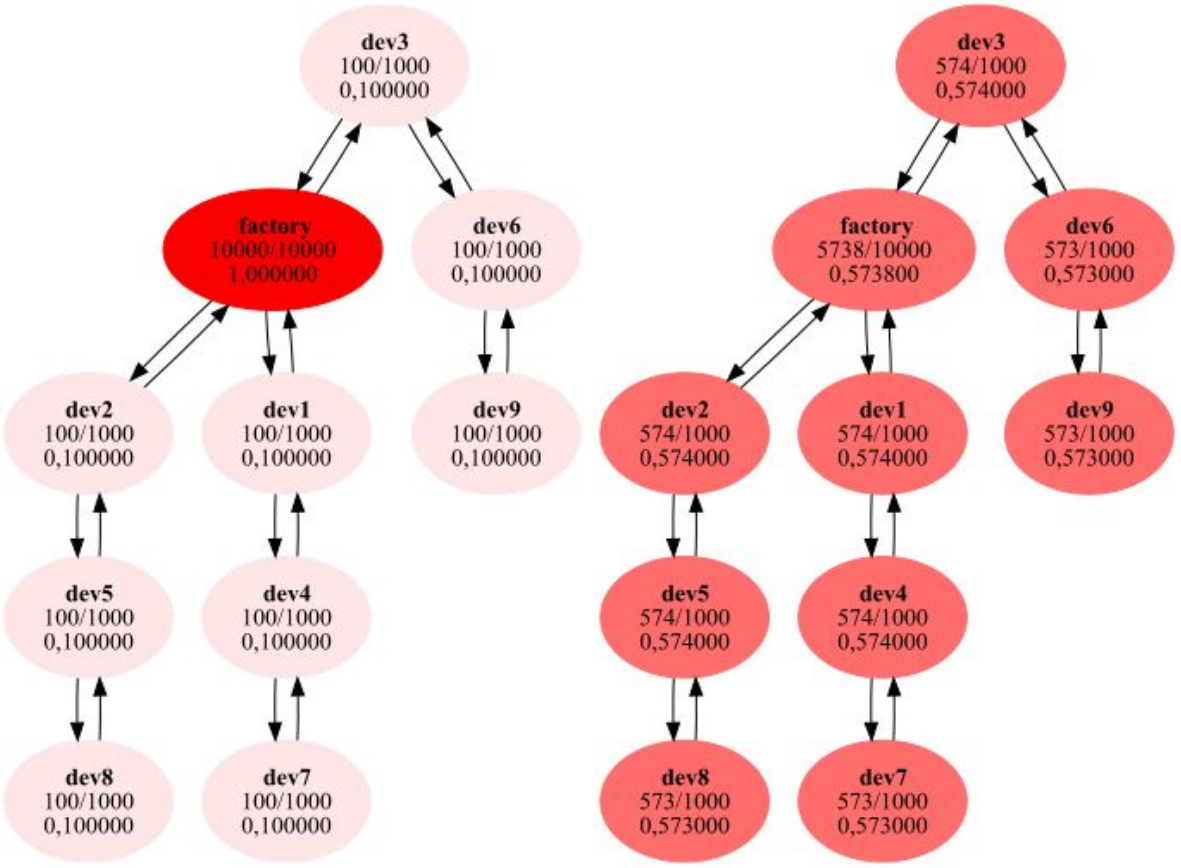


- Scenariusz 3 (równowaga po 16048s)





- Scenariusz 4 (równowaga po 28454s)



3. Wnioski

Przygotowany algorytm pozwala na zrównoważenie stężeń, niezależnie od przygotowanej konfiguracji. Wolne tempo równoważenia stężeń ma miejsce ze względu na to, że zadania są przesyłane pojedynczo, bez względu na różnicę stężeń między dwoma urządzeniami.

4. Linki

1. Repozytorium projektu na platformie GitHub:
   https://github.com/artudi54/Osmosis

2. Graphviz, oprogramowanie do tworzenia grafów:
   https://github.com/nidi3/graphviz-java

3. Kod wykorzystany do tworzenia gifów:
   https://gist.github.com/jesuino/528703e7b1974d857b36