Metody projektowania infrastruktury telekomunikacyjno-obliczeniowej
23Z

Jakub Górczyński, Marcin Ziółkowski

Zdefiniowanie Problemu

Celem zadania projektowego było zaprojektowanie algorytmu orkiestracji usług dla systemu MEC (obliczeń na brzegu sieci), który zapewni efektywne wykorzystanie zasobów w przypadku oferowania usług złożonych z mikrousług. Do implementacji wykorzystano narzędzie Minizinc oraz przetestowano kod źródłowy pod kątem znalezienia najlepszego rozwiązania przy użyciu różnych solverów.

Formalizacja problemu

Oznaczenia

Oznaczenie	Wartości	Opis
Dane wejściowe		
Węzły		
V	$\{v_1, v_2, \ldots, v_{\overline{v}}\}$	Zbiór węzłów, w których można alokować aplikacje
$\overline{V}$	=|V|	Liczba węzłów
C	$\{c_1, c_2, \ldots, c_{\overline{v}}\}$	Zbiór maksymalnych pojemności per węzeł
D	$D \epsilon Z^+$	Opóźnienie pomiędzy węzłami
T	$T \epsilon Z^+$	Koszt transmisji pomiędzy węzłami
Alokowane aplikacje		
K	$\{UPF, APP\}$	Zbiór typów aplikacji alokowanych w systemie
R_A	$A \epsilon Z^+$	Liczba żądań użytkowników per węzeł
D_{max}	$D_{max} \epsilon Z^+$	Całkowite opóźnienie per aplikacja
Zmienne decyzyjne		
$x_{V,K}$	$x_{V,K} \epsilon Z^+$	Zmienna opisująca alokację: węzeł (V), na którym jest UPF lub aplikacja (K)

$y_{A,K,V,V}$	$y_{A,K,V,V} \epsilon \{0, \dots, 1000\}$	Zmienna opisująca rozpływ ruchu: aplikacja typu K na podstawie źródła żądania A powinna przesłać swój ruch do UPF uruchomionego na węźle V, i powinna zostać uruchomiona na węźle nr V. Maksymalna wartość powinna być ustawiona przez dostawcę usług, i może być inna dla danego typu aplikacji.

Ograniczenia

Ograniczenie (1) zapewnia spełnienie **warunku na rozpływ ruchu** – napływający ruch zostanie przekierowany do odpowiednich węzłów, dzięki czemu cały ruch będzie obsłużony:

$$\sum_{v \epsilon V} \sum_{u \epsilon V} y_{a,k,v,u} \geq R_a; \forall k \epsilon K, a \epsilon A, k \neq UPF \quad (1)$$

Ograniczenie (2) zapewnia, że sumaryczny ruch wychodzący z węzła musi być mniejszy lub równy względem zdolności obliczeniowej, którą opisane są uruchomione na węźle aplikacje.

$$\sum_{a \epsilon A} \sum_{v \epsilon V} y_{a,k,v,u} \leq x_{V,APP}; \forall u \epsilon V, k \epsilon K, k \neq UPF \quad (2)$$

Ograniczenie (3) zapewnia, że suma ruchu idącego przez węzeł musi być mniejsza lub równa względem zdolności obliczeniowej, którą oferuje dana liczba funkcji UPF uruchomionych na tym węźle:

$$\sum_{a \epsilon A} \sum_{k \epsilon K} \sum_{u \epsilon V} y_{a,k,v,u} \leq x_{V,UPF}; \forall v \epsilon V \quad (3)$$

Ograniczenie (4) zapewnia, że zasobów oferowanych przez węzły jest więcej niż liczba uruchomionych na nich funkcji UPF i aplikacji:

$$C_v \geq x_{V,UPF} + x_{V,APP}; \forall v \epsilon V \quad (4)$$

Ograniczenie (5) zapewnia, że sumaryczna wartość opóźnienia dla całej drogi, na której przesyłany jest ruch, jest mniejsza, niż jest to wyrażone w żądaniu:

$$D_{a,v} + D_{v,u} \leq D_{max}; \forall \begin{cases} a \epsilon A, \\ k \epsilon K, \\ v \epsilon V, u \epsilon V, y_{a,k,v,u} > 0 \end{cases} \quad k \neq UPF \tag{5}$$

Funkcja celu

Celem algorytmu rozmieszczania aplikacji jest minimalizacja sumy kosztów alokacji. Funkcję celu wyraża równanie (8), które *de facto* wynika z sumy ograniczeń (4) i (5), opisujące odpowiednio całkowity koszt rozmieszczenia (6) i całkowity koszt transmisji (7):

$$g = \sum_{v \epsilon V} x_{V,UPF} + x_{V,APP}\,; \forall v \epsilon V \tag{6}$$

$$h = \sum_{a \epsilon A} \sum_{k \epsilon K} \sum_{v \epsilon V} \sum_{u \epsilon V} T_{a,v} + T_{v,u}; k \neq UPF, y_{a,k,v,u} > 0 \tag{7}$$

$$solveminimize f = g + h \tag{8}$$

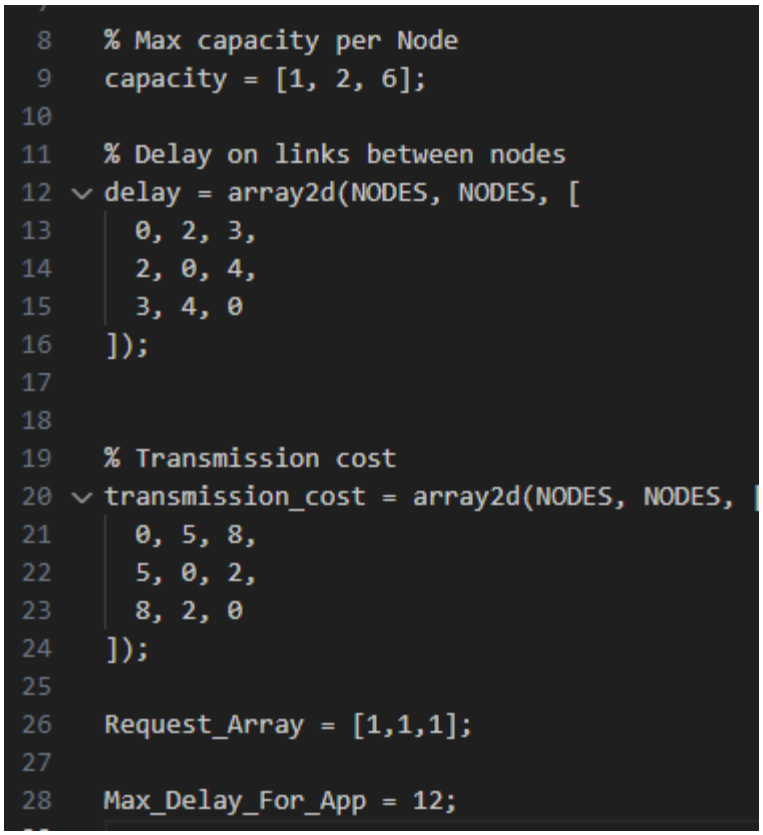
Założenia

W naszym modelu przyjęliśmy następujące założenia:

- Istnieje tylko 1 typ aplikacji, który ma takie samo wymaganie dotyczące maksymalnego opóźnienia dla każdego żądania. Model można rozszerzyć o wiele innych typów aplikacji z innymi akceptowalnymi wartościami dla opóźnienia.
- Przesyłanie danych między węzłami ma swój koszt będący częścią funkcji celu, ale nie wpływa na zużycie zasobów w węzłach.
- Z uwagi na kompromis między złożonością danych wprowadzanych do modelu a czasem znalezienia rozwiązania, będącym zależnym od poziomu złożoności problemu, w modelu przyjęliśmy koszt zasobów uruchomienia UPF i aplikacji jako 1, oraz przyjęliśmy po 1 żądaniu wychodzącym z każdego węzła. Również zdolność obsługowa dla UPF i aplikacji wynosiła 1, tak samo jak wolumen każdego nadchodzącego żądania.

Dane

Na potrzeby testów modelu przygotowaliśmy dane w następujących formatach: 3x3, 5x5, 6x6 i 7x7. Oznacza to, że w formacie 3x3 są 3 węzły i 3 żądania użytkowników, które przychodzą do konkretnych węzłów.



```
% Max capacity per Node
capacity = [1, 2, 6];

% Delay on links between nodes
delay = array2d(NODES, NODES, [
  0, 2, 3,
  2, 0, 4,
  3, 4, 0
]);


% Transmission cost
transmission_cost = array2d(NODES, NODES, [
  0, 5, 8,
  5, 0, 2,
  8, 2, 0
]);

Request_Array = [1,1,1];

Max_Delay_For_App = 12;
```

Dane wprowadzane w formacie 3x3

`Capacity` to tablica zawierająca w sobie maksymalne zasoby dla każdego węzła.

`Delay` to dwuwymiarowa tablica mówiąca o opóźnieniu, jakie wprowadzi przesłanie danych z danego węzła do innego węzła. Przykład: delay[1,3] = 8, co oznacza, że opóźnienie między 1 a 3 węzłem wynosi 8.

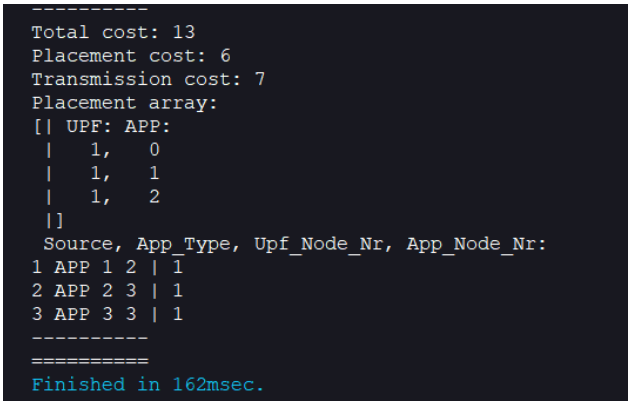
`Transmission cost` analogicznie do `Delay`, zawiera koszt przesłania danych między węzłami.

`Request Array` zawiera w sobie liczbę żądań użytkowników dla danego węzła (oznaczony indeksem).

`Max_Delay_For_App` to maksymalna wartość opóźnienia tolerowana dla danego typu aplikacji.

Podczas walidacji modelu testowaliśmy również sytuacje, w których przychodziło wiele żądań do jednego węzła oraz kiedy koszt uruchomienia aplikacji na węzłach był różny (w celu odzwierciedlenia rozmieszczenia geograficznego węzłów).

Walidacja modelu



Dla formatu danych 3x3, zawartego w sekcji powyżej uzyskano następujące rozwiązanie.

Sumaryczny koszt to 13 (6 z racji uruchomienia 3 UPF i 3 aplikacji) oraz 7 z racji przesłania danych (UPF -> Aplikacja -> Źródło żądania). Wartość zmiennej X oznacza, że na węźle 3 został uruchomiony 1 UPF i 2 instancje aplikacji.

Następnie widzimy wartość zmiennej y, mówiącej o rozpływie ruchu. Dla y [1, APP, 1, 2] wartość 1 oznacza, że dla aplikacji uruchomionej na węźle nr 2, która wysyła ruch do UPF na węźle o nr 1, który następnie przesyła ruch do węzła o nr 1 (źródło żądania - ten sam węzeł, na którym jest uruchomiony UPF, więc darmowy przesył). Tym samym w tym przypadku mamy następujące koszty przesłania danych: Z węzła 3 do 2, oraz z węzła 2 do 1.

```
% Transmission cost
transmission_cost = array2d(NODES, NODES, [
  0, 5, 8,
  5, 0, 2,
  8, 2, 0
]);
```

Koszty transmisji danych

Koszt przesłania z nr 3 do 2 wynosi 2 (3 wiersz 2 kolumna) I analogicznie z 2 do 1 oznacza koszt 5. Sumarycznie otrzymujemy koszt transmisji dla całego rozwiązania równy 7, co zgadza się z uzyskanym przez nas wynikiem.

Porównanie solverów

Środowisko uruchomieniowe MiniZinc IDE w wersji 2.7.6 oferuje szereg dostępnych solverów:

Nazwa solvera	**Wersja oprogramowania**
Chuffed	0.12.1
COIN-BC	2.10.10/1.17.8
Gecode	6.3.0
Gecode Gist	6.3.0
Globalizer	0.1.7.2
HiGHS	1.5.1

Gecode Gist oraz Globalizer wymagały dodatkowej konfiguracji lub próbowały wizualizować problem, dlatego przeprowadziliśmy pomiary dla pozostałych 4 solverów.

W ramach porównania przyjęto dwa kryteria: **czas działania** oraz **zwracany wynik funkcji kosztu**.

Chuffed 0.12.1	3x3	5x5	7x7
Średni najlepszy wynik	13	12	21
Średni czas szukania [msec]	278,9	895,2	464000

Wyniki uzyskane za pomocą Solvera Chuffed 0.12.1

COIN-BC 2.10.10	3x3	5x5	7x7
Średni najlepszy wynik	13	12	21
Średni czas szukania [msec]	160,8	197,7	436,2

Wyniki uzyskane za pomocą COIN-BC 2.10.10

Geocode 6.3.0	3x3	5x5	7x7
Średni najlepszy wynik	13	12	21
Średni czas szukania [msec]	187,5	217,8	DNF

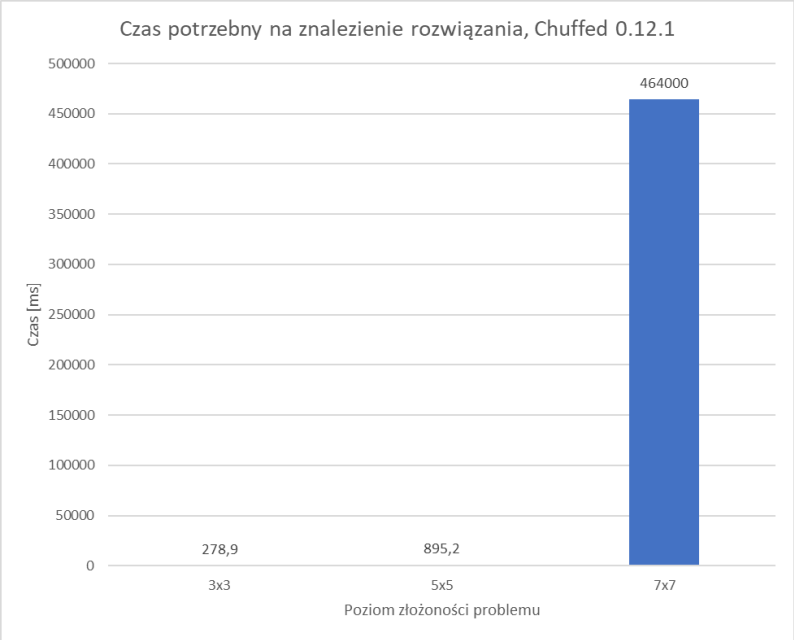
Wyniki uzyskane za pomocą COIN-BC 2.10.10

HIGHS 1.5.1	3x3	5x5	7x7
Średni najlepszy wynik	13	12	21
Średni czas szukania [msec]	163,5	182,2	268,5

Wyniki uzyskane za pomocą COIN-BC 2.10.10

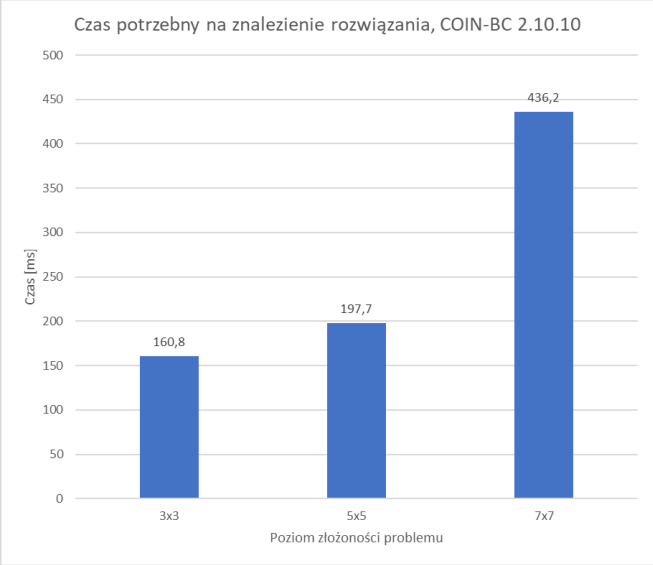
Ponieważ wszystkie solvery zwróciły tak samo dobry wynik (pod względem kryterium najmniejszej wartości funkcji celu, łącznego kosztu) w tej sekcji skupimy się na czasie, jaki był potrzebny na znalezienie rozwiązania.

Chuffed 0.12.1



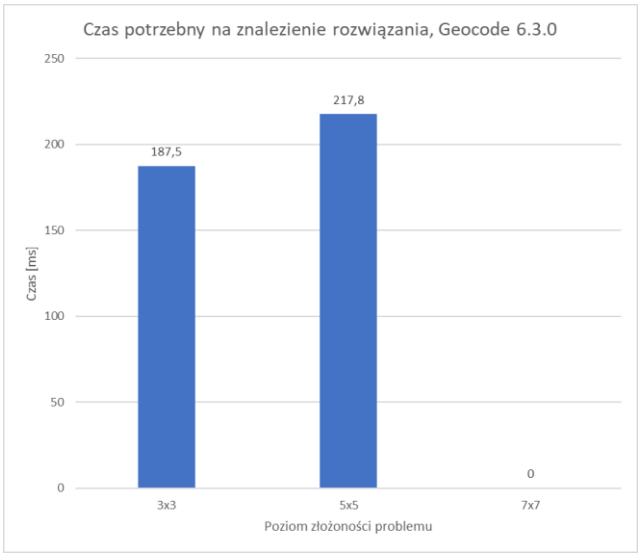
Dla solvera Chuffed 0.12.1 możemy zaobserwować ponad trzykrotny wzrost dla danych wejściowych składających się z 5 węzłów w porównaniu do 3 węzłów. Rozwiązanie dla danych 7x7 zajęło nam ponad 7 minut, czyli 500 razy dłużej, niż dla modelu 5x5. W porównaniu do innych solverów, Chuffed 0.12.1 wydaje się niezdatny do pracy z bardziej złożonymi problemami. Nawet dla danych 3x3, solver ten uzyskał najgorszy czas z wszystkich testowanych solverów.

COIN-BC 2.10.10



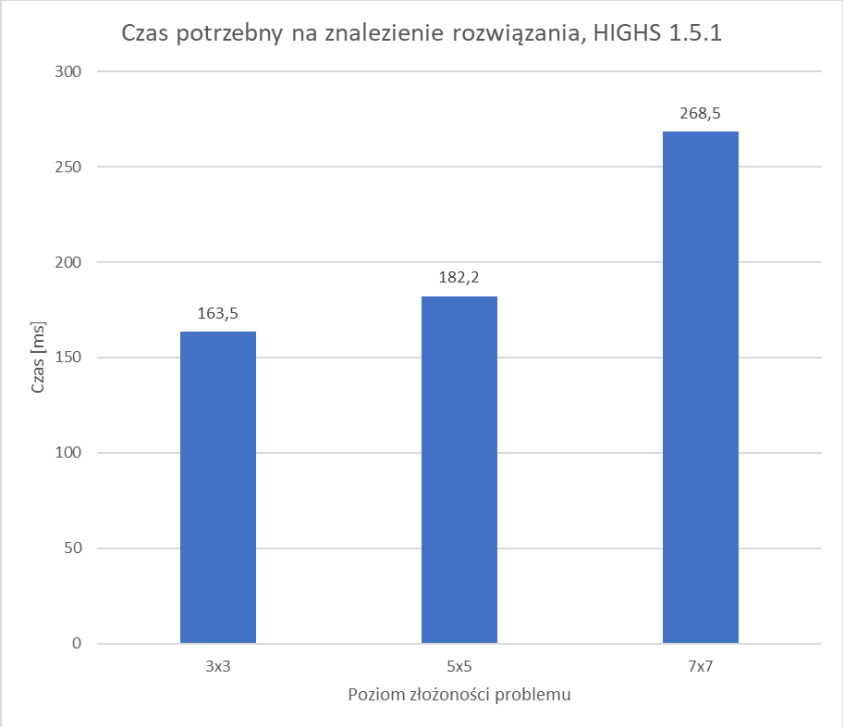
COIN-BC 2.10.10 najszybciej znalazł rozwiązanie dla problemu o danych wejściowych w formacie 3x3. W formacie 5x5 uzyskał drugie najlepsze miejsce, tak samo dla formatu 7x7, gdzie znalezienie rozwiązania zajęło 436 milisekund, czyli zaobserwowaliśmy wzrost o 132% w porównaniu do formatu 5x5.

Gecode 6.3.0



Solver Gecode dość szybko poradził sobie z problemami w formacie 3x3 I 5x5 (zajął odpowiednio 3 I 3 miejsce wśród 4 testowanych problemów, jednak tutaj różnica jest na poziomie 20-25 milisekund), jednak jest to jedyny solver który nie znalazł rozwiązania dla problemu w formacie 7x7 mimo wielokrotnych prób o długosci kolejno 15 minut, godziny a nawet 2.5 godziny. Tym samym solver Gecode w wersji 6.3.0 również naszym zdaniem nie jest w stanie poradzić sobie z zbyt skomplikowanymi problemami.

HIGHS 1.5.1



Solver HIGHS najszybciej znalazł rozwiązanie dla formatów 5x5 i 7x7, w formacie 3x3 różnica względem COIN-BC wynosiła niecałe 3 milisekundy. Solver HIGHS sprawdził się zdecydowanie najlepiej z testowanych przez nas solverów. W formacie 7x7 obserwujemy tylko 47.3% wzrost w czasie potrzebnym na znalezienie rozwiązanie w porównaniu z formatem 5x5. Jest to najmniejszy wzrost w przypadku tych dwóch formatów i na bazie tych pomiarów solver HIGHS wydaje się najbardziej obiecujący przy rozwiązywaniu złożonych problemów.