AGH Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Dokumentacja

Kamil Potoczny, Bartosz Piwnik

Systemy równoległe i rozproszone - aplikacja Java RMI

Monte Carlo - ruch na rondzie

Kraków, Czerwiec 2016
AGH, WFiIS, Informatyka Stosowana

1 Wstęp

Celem niniejszego projektu było stworzenie aplikacji symulującej ruch pojazdów na rondzie z wykorzystaniem metody Monte-Carlo.

Zastosowany algorytm został oparty na podejściu zaprezentowanym w książce: Michael J. Quinn, "Parallel Programing in C with MPI and OpenMPI".

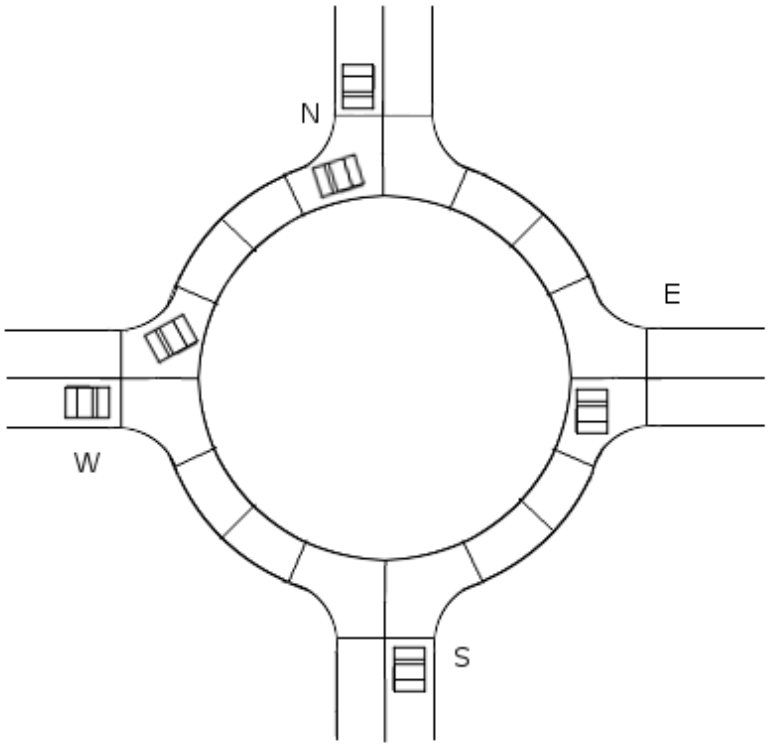
Implementacja została zrealizowana w języku Java, przy wykorzystaniu standardu RMI. Aplikacja została napisana z użyciem architektury klient-serwer. Część serwerowa odpowiada za wykonanie potrzebnych obliczeń, a część kliencka za ich pobranie i wyświetlenie. Możliwe jest uruchomienie wielu instancji serwera na różnych portach, a połączenie z każdą z nich poprzez klienta realizowane jest w osobnych wątkach.

Aplikacja wykorzystuje funkcjonalności Java 8, dlatego jest ona konieczna do kompilacji i uruchomienia.

2 Działanie algorytmu

Dla celów symulacji zakładamy tradycyjne założenia dotyczące ruchu na rondzie: samochody poruszają się w kierunku przeciwnym do ruchu wskazówek zegara, natomiast pojazdy znajdujące się obecnie na rondzie mają pierwszeństwo przed pojazdami wjeżdżającymi na nie.

Główny obszar reprezentujący rondo, podzielony jest na 16 segmentów (podczas pojedynczej iteracji wszystkie auta przemieszczają się o jedną komórkę), występują także 4 możliwe zjazdy/wjazdy na rondo (oznaczone, jako N, S, W, E). Całość przedstawia poglądowo poniższy schemat:



Rysunek 1Rondo z czterema wjazdami, podzielone na 16 sekcji.

Pojazd, który chce wjechać na rondo, może dokonać tego manewru tylko i wyłącznie wtedy, gdy żaden inny samochód poruszający się obecnie po rondzie nie próbuje przemieścić się do tej samej komórki.

Do symulacji wykorzystane są 2 podstawowe struktury danych:

- tablica f, zawierająca średnie odstępy czasowe pomiędzy nadjeżdżającymi pojazdami dla każdego z 4 wjazdów
- macierz d o rozmiarze 4x4, w której poszczególne komórki przedstawiają prawdopodobieństwo, iż samochód wjeżdżający na rondo wjazdem „i" opuści go zjazdem „j"

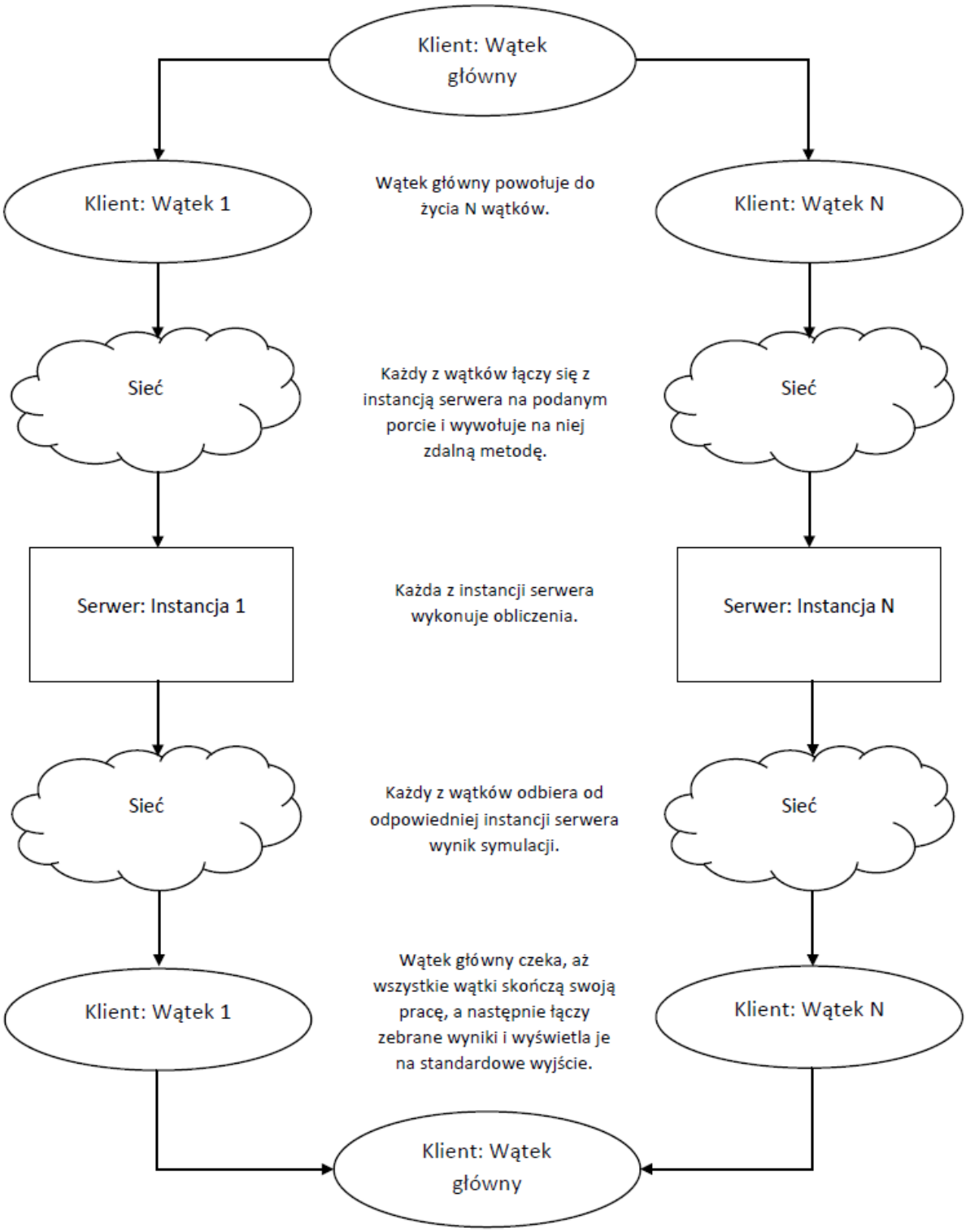
Przebieg każdej iteracji programu może zostać podzielony na 3 charakterystyczne kroki:

- Dotarcie nowych pojazdów do ronda
- Samochody znajdujące się już na rondzie przemieszczają się do kolejnej komórki. Pojazdy, które docierają do swojego zjazdu opuszczają skrzyżowanie, a ich ruch nie jest już dalej symulowany.
- Po przemieszczeniu samochodów algorytm sprawdza, czy komórka odpowiadająca danemu wjazdowi jest pusta. Jeśli tak, na rondo wpuszczane są pojazdy oczekujące w kolejce, bądź w razie pustej kolejki, te, które dopiero docierają do skrzyżowania. Jeśli wjazd na rondo nie jest możliwy, samochody dodawane są do kolejki oczekujących.

Rezultatem programu będzie statystyka dla każdego z 4 wjazdów/zjazdów, udzielająca odpowiedzi na 2 zasadnicze pytania:

- Jakie jest prawdopodobieństwo sytuacji, iż samochód docierający do ronda będzie musiał oczekiwać na wjazd
- Jaka jest średnia długość kolejki oczekujących pojazdów

Rozproszenie w naszym programie zostało zrealizowane w następujący sposób: na każdym z serwerów uruchamiana jest osobna symulacja ruchu o takiej samej liczbie iteracji. Po zakończeniu symulacji na wszystkich instancjach, klient zbiera informacje i łączy wyniki. Takie działanie może zostać zilustrowane następującym schematem:



Rysunek 2. Przedstawia schemat działania aplikacji klient-serwer, gdzie N to ilość uruchomionych serwerów/wątków

3 Format danych wejściowych

Wspomniane wcześniej tablice średnich czasów i prawdopodobieństw (f i d), wczytywane są do programu z plików wejściowych. Aby działanie aplikacji było poprawne należy upewnić się, iż poniższy format danych jest zachowany:

- Plik input_f.dat - zawiera tablicę średnich czasów f, gdzie poszczególne komórki oddzielone są spacjami. Tablica składa się dokładnie z 4 elementów, po jednym dla każdego z wjazdów
- Plik input_d.dat - zawiera macierz o wymiarze 4x4, gdzie poszczególne komórki wiersza oddzielone są spacjami

Przykładowy format danych wejściowych dla pliku input_f.dat:

```
3 3 4 2
```

Przykładowy format danych wejściowych dla pliku input_d.dat:

```
0.1 0.2 0.5 0.2
0.2 0.1 0.3 0.4
0.5 0.1 0.1 0.3
0.3 0.4 0.2 0.1
```

4 Obsługa aplikacji

Do obsługi programu przygotowany został plik Makefile oferujący następujące komendy:

- make classes - kompiluje program
- make runServer – uruchamia 4 instancje serwerów na portach: 1097, 1098, 1198, 1234
- make runClient – uruchamia klienta, który łączy się z serwerami na portach 1097, 1098, 1198, 1234 portach i wykonuje na nich obliczenia
- make clean - przywraca katalog do stanu wyjściowego

Istnieje możliwość zmiany portów, na których uruchamiany jest serwer i z których korzysta klient, poprzez modyfikację pliku makefile lub uruchomienie programu bezpośrednio poprzez komendy:

- java server.RMIServer {ports} – uruchamia serwer na podanych portach
- java client.RMIClient {ports} – uruchamia klienta korzystającego z podanych portów

Trzeba jednak pamiętać, że należy podać porty, do których mamy dostęp poprzez technologię RMI oraz które nie są zajęte.