Badanie porównawcze - metody synchronizacji wątków

Szymon Twardosz - grupa 11

29 listopad 2023

1 Wprowadzenie

1.1 Cel

Eksperyment polegał na zmierzeniu stopnia zagłodzenia, oraz szybkości zaimplementowanych metod synchronizacji wątków w Javie. Problemem branym pod uwagę podczas synchronizacji wątków był problem producenta-konsumenta. W ramach badania zaimplementowane zostały klasy:

1. **FourConditionBuffer** - rozwiązanie problemu na czterech zmiennych Condition oraz jednej zmiennej Lock
2. **ThreeLockBuffer** - rozwiązanie problemu na trzech zmiennych Lock i jednej zmiennej Condition
3. **TwoConditions** - rozwiązanie problemu na dwóch zmiennych Conditions i jednej zmiennej Lock

Załączone wykresy to pojedyncze przykłady przeprowadzonych testów. W rzeczywistości dla jednego zestawu parametrów przeprowadzone zostało kilka testów.

1.2 Dane techniczne

Doświadczenie realizowane było w języku Java wersji 17 oraz środkowiska IntelliJ. Sprzęt doświadczalny to laptop z systemem operacyjnym Windows 10 x64, o 4-rdzeniowym procesorze AMD Ryzen 7 3750 2.30GHz, z zainstalowaną pamięcią RAM 8,00GB

2 Opis metod

2.1 FourConditionBuffer - 1 Lock 4 Conditions

Metoda posiada dwa rodzaje kolejek:

- First - kolejka dla wątków (danego typu), które pierwsze znalazły się w buforze (wtedy kolejka First, była pusta).
- Other - kolejka dla wątków (danego typu), do której trafiają wątki gdy w kolejce First znajduje się już inny wątek.

Wątek wywołujący metodę na buforze zachowuję się w określony sposób. Jeżeli kolejka First jest pusta, to wątek otrzymuje pierwszeństwo nad pozostałymi konkurentami. Oznacza to, że każdy następny obiekt, który będzie próbował wywołać tą samą metodę na buforze, będzie zawieszany na kolejce Other. Następnie będzie czekał dopóki, wątek z kolejki First nie wykona swojej operacji w buforze. Przy opuszczaniu bufora, każdy obiekt budzi, jeden wątek z kolejki Other, oraz z kolejki First (przeciwnego typu).

2.2 ThreeLockBuffer - 3 Locki 1 Condition

Metoda, która posiada jedną wspólną zmienną Lock oraz dwie zmienne Lock dla poszczególnych klas wątków. Na początku wątek zdobywa swojego klasowego Locka. Następnie czeka na możliwość wejścia do wspólnego Lock'a. Później oczekuje na możliwosć konsumpcji/produkcji (jeżeli nie jest w stanie tego zrobić). Po operacji na buforze „budzi"wątek przeciwnego typu czekający na Condition.
System ten ma zagwarantować brak zagłodzenia. W związku z tym, że po zdobyciu Lock'a danego typu, wątek posiada „pierwszeństwo"przed pozostałymi wątkami swojej klasy. Jest to spowodowane faktem, iż daną zmienną Lock może w jednym momencie posiadać tylko jeden wątek. Następnie o wspólny Lock konkurują maksymalnie dwa wątki o przeciwnych typach.

2.3 TwoConditons - 1 Lock 2 Conditions

Metoda, która nie gwarantuje brak zagładzania wątków. Posiada dwie kolejki Conditions (po jednej dla każdego typu wątków). Czekają w nich wątki, które po wejściu do metody bufora nie mogły wykonać swojej funkcji ze względu na aktualną ilość elementów w monitorze.

3 Liczba produkcji/konsumpcji w określonym czasie

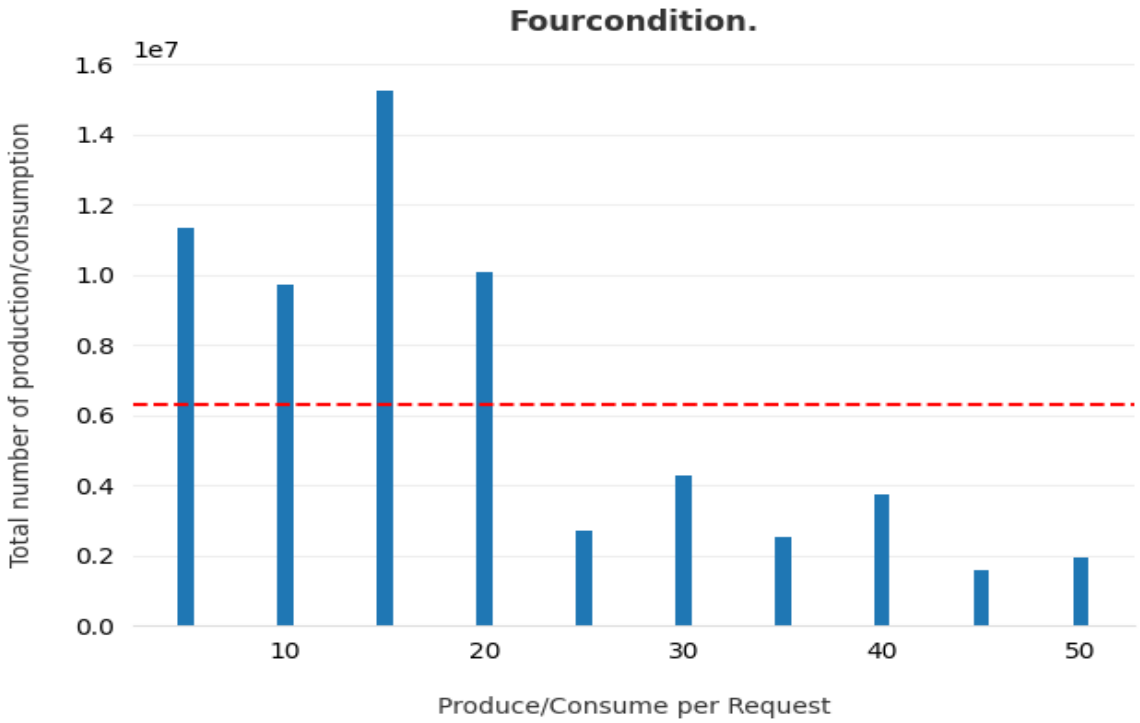
3.1 Opis

Eksperyment, którego celem jest obserwacja braku zagładzania wątków na poszczególnych buforach. Polega on na pomiarze liczby konsumpcji/produkcji jakie wykonuję każdy wątek. Ponadto, każdy z zainicjalizowanych wątków posiada określoną (z góry przypisaną) liczbę jaką wysyła do bufora.W związku z tym w systemie działają wątki zarówno „chciwe"jak i te mało wymagające.
Dla każdego pomiary należy wyodrębnić użyte parametry. Są to:

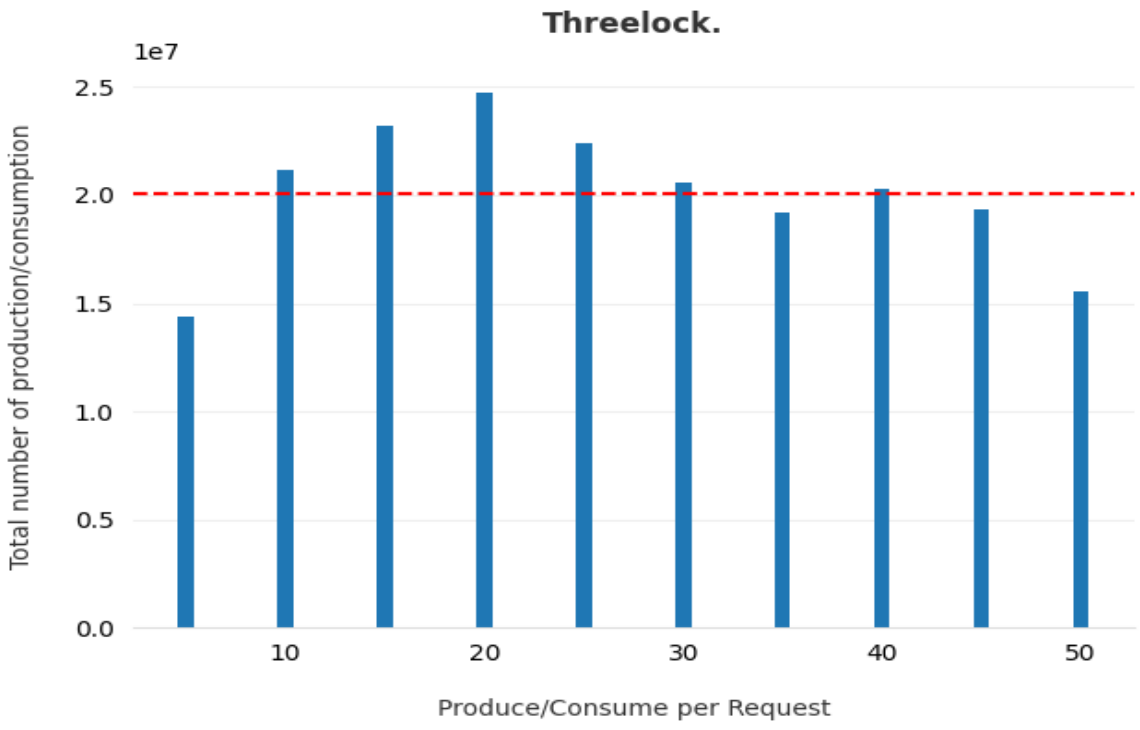
- Czas trwania testu dla pojedynczego bufora
- Pojemność bufora
- Liczba wątków producenta i konsumenta
- Maksymalna liczba wysłana do bufora (maksymalna prośba o produkcję/konsumpcję)

3.2 Wyniki

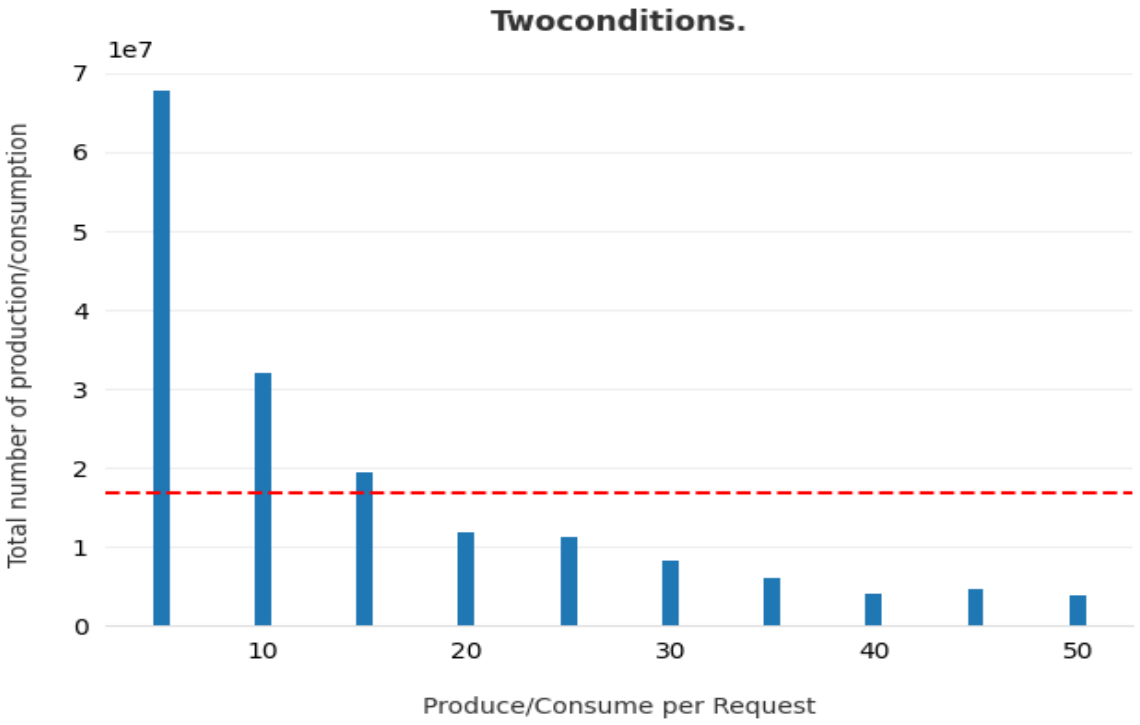
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 7
  Maksymalna liczba wysyłana do Bufera: 50



Rysunek 1: Wyniki dla FourConditionBuffer



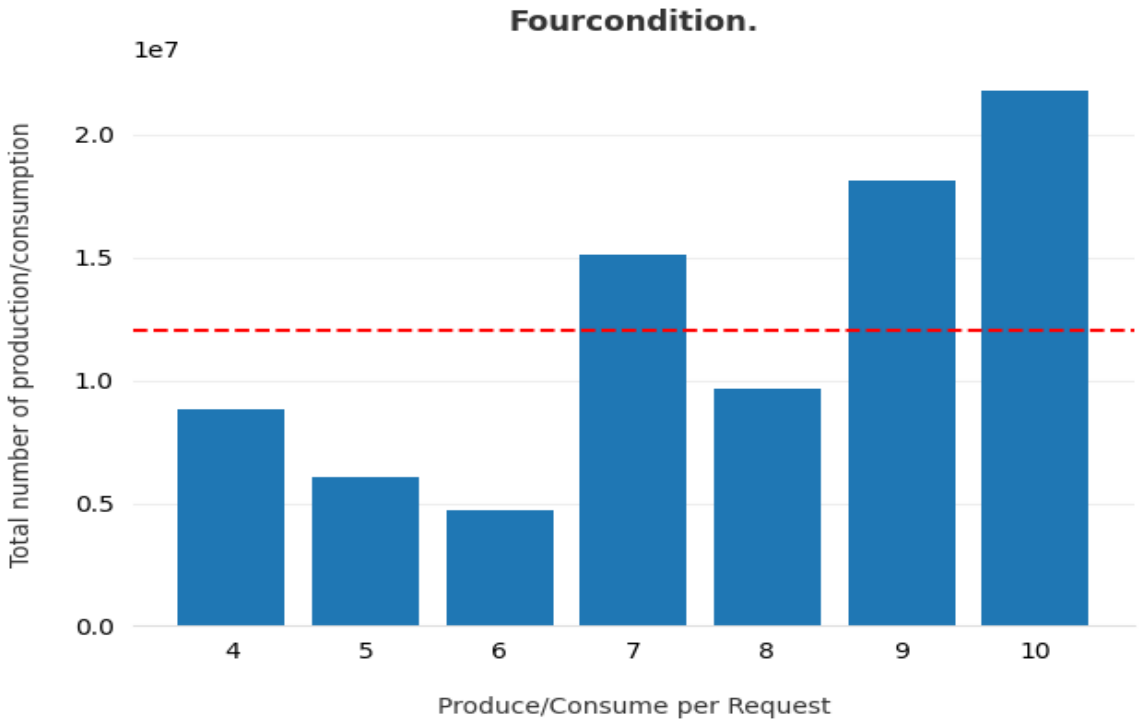
Rysunek 2: Wyniki dla ThreeLockBuffer



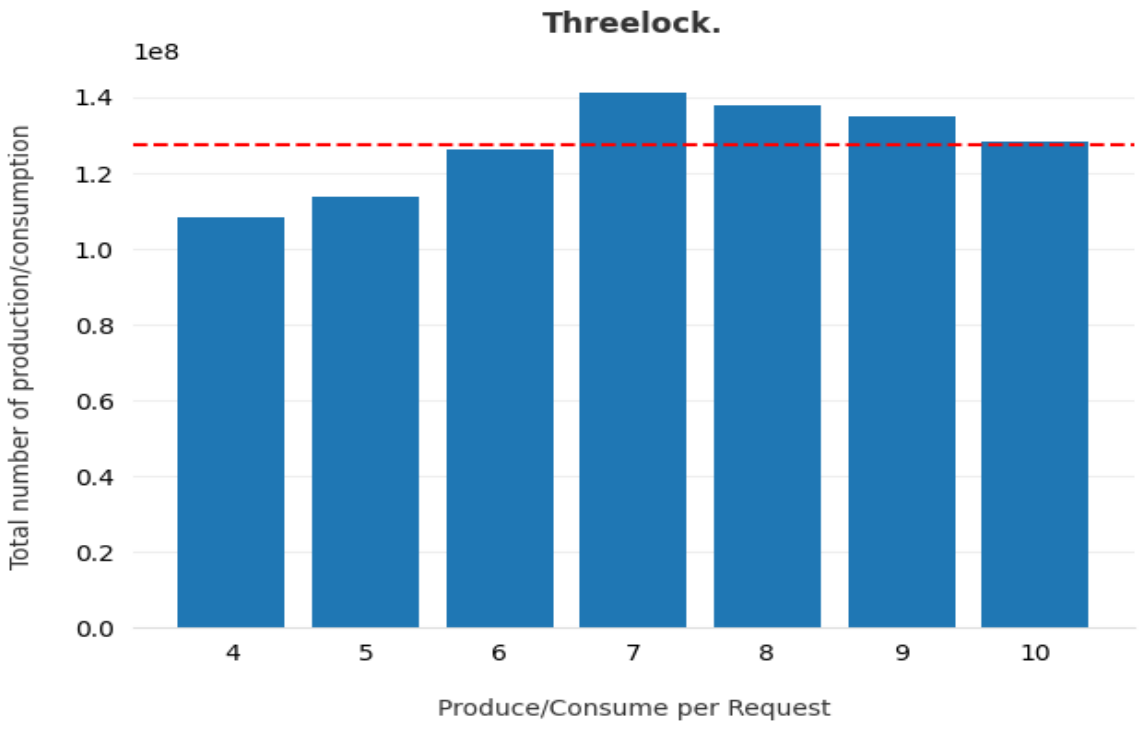
Rysunek 3: Wyniki dla ThreeLockBuffer

Wnioski Zaimplementowane metody (fourConditons i threeLock) nie powinny zagładzać wątków. Mimo to, dla implementacji na czterech zmiennych Conditions niektóre z nich są zagładzane. „Chciwe"wątki o wiele rzadziej wykonują operację na buforze. Czy zmiana parametru maxRequest (maksymalna liczba wysyłana do bufora) polepszy sytuację?
Z drugiej strony, metoda twoConditions zgodnie z przypuszczeniami zagładza wątki bardziej wymagające.

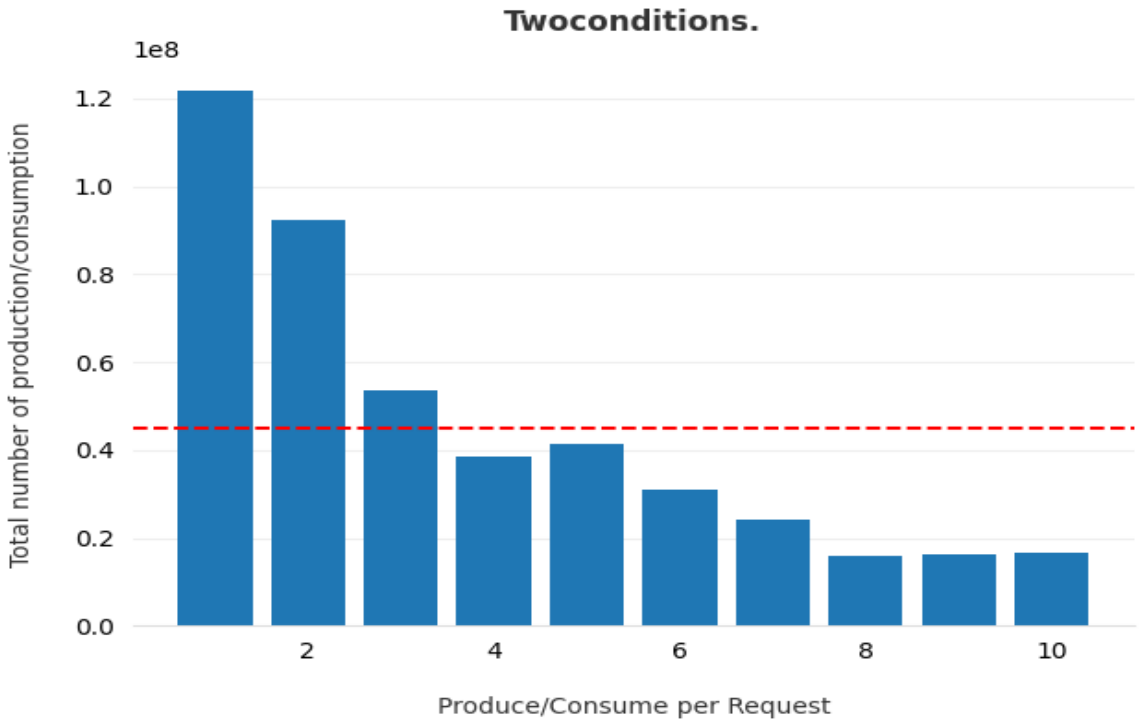
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 10



Rysunek 4: Wyniki dla FourConditionBuffer



Rysunek 5: Wyniki dla ThreeLockBuffer



Rysunek 6: Wyniki dla ThreeLockBuffer

Wnioski Metoda oparta na trzech zmiennych Lock nadal radzi sobie z zagładzaniem. Natomiast, implementacja FourCondition, mimo „łagodniejszych"parametrów wciąż zagładza wątki. Tym razem zagładzanymi wątkami są te o mniejszych wymaganiach. W takim razie, w tej sytuacji powodem zagładzania może nie być implementacja bufora, ale sposób działania scheduler'a systemowego. Ostatnia metoda - oparta na dwóch Conditions, wciąż zagładza wątki o większych wymaganiach.

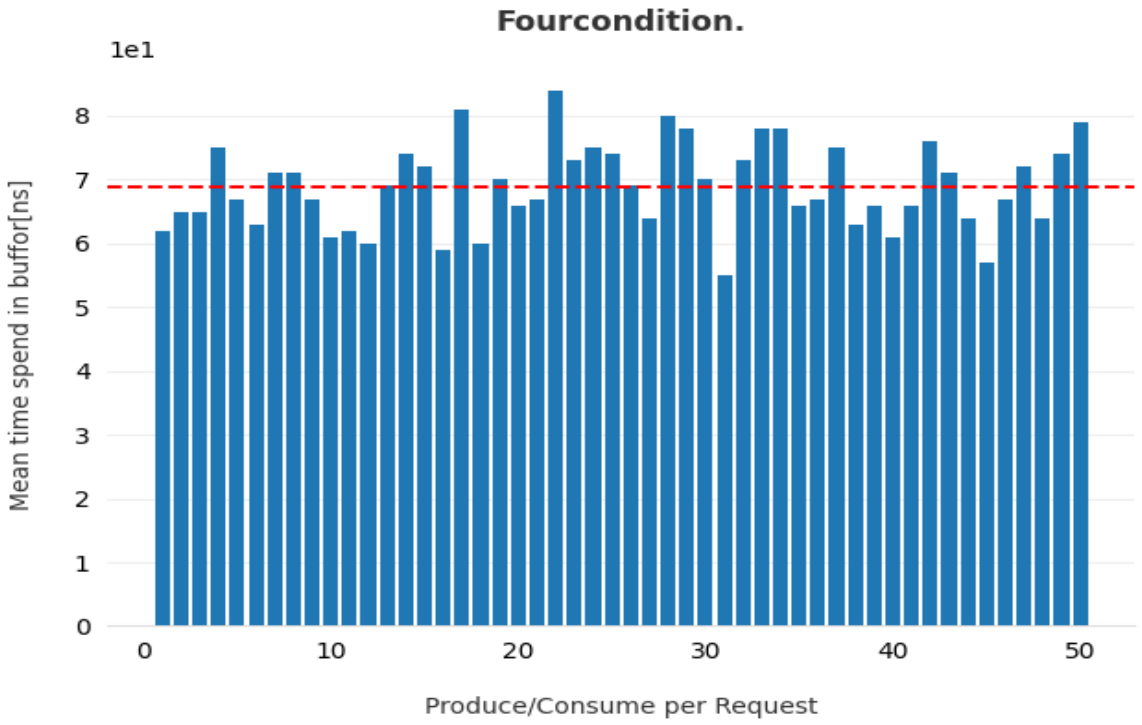
4 Czas spędzony w Buforze

4.1 Opis eksperymentu

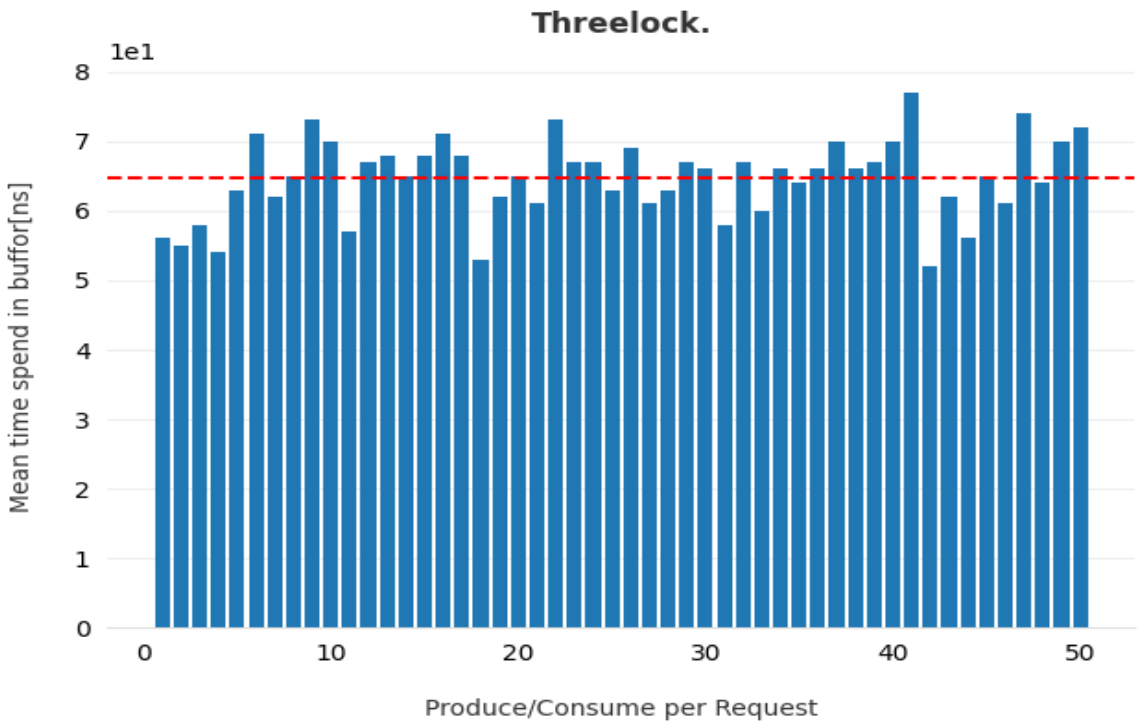
Eksperyment polegał na zmierzeniu średniego czasu, jaki zajmuję w buforze wątek, o określonej produkowanej/konsumowanej części. Jego celem jest sprawdzenie czy dany bufor (jego implementacja w Javie) faktycznie nie zagładza wątków. Każdy wątek działa w nieskończonej pętli. Za każdym razem, przed wejściem do bufora losuję określoną liczbę elementów. Następnie włączany jest zegar, który mierzy ile czasu dany wątek spędzi w buforze. Wątek wywołuje metodą consume()/produce(). Następnie zegar jest zatrzymywany i czas zapisywany jest w odpowiednie miejsce w tablicy wyników. Na końcu dla każdej liczby elementów, liczony jest średni czas jaki wątek musiał spędzić w buforze.
Parametry pomiarów są takie same jak w poprzednim eksperymencie.

4.2 Wyniki

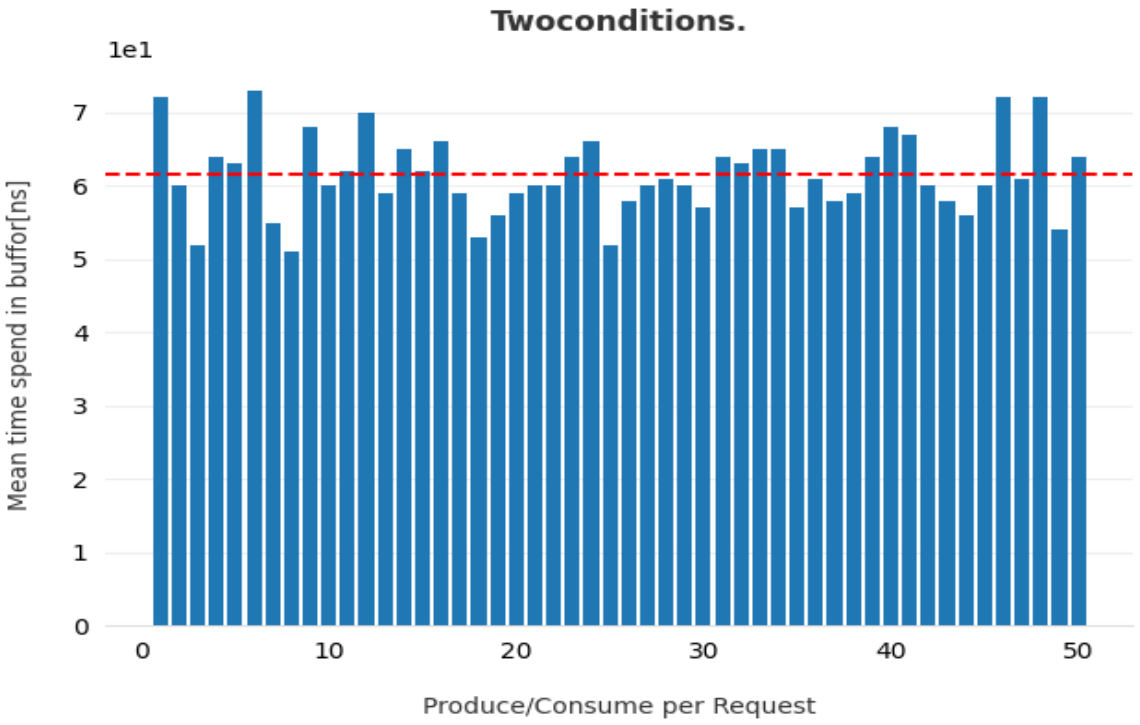
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 50



Rysunek 7: Wyniki dla FourConditionBuffer



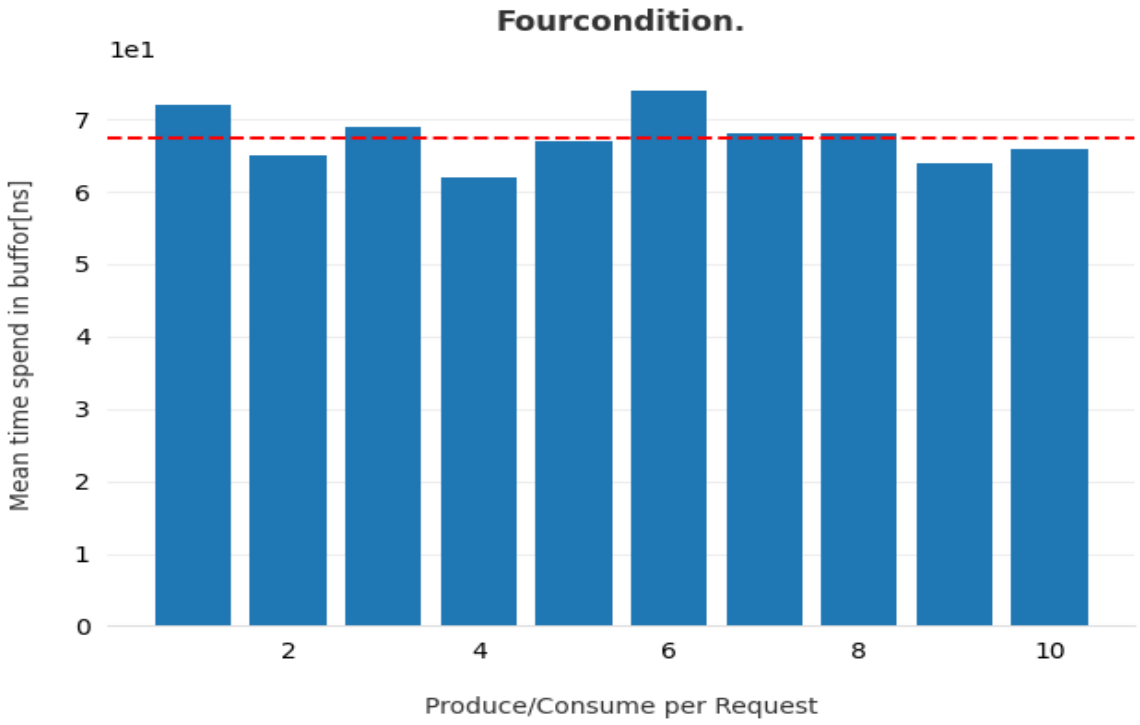
Rysunek 8: Wyniki dla ThreeLockBuffer



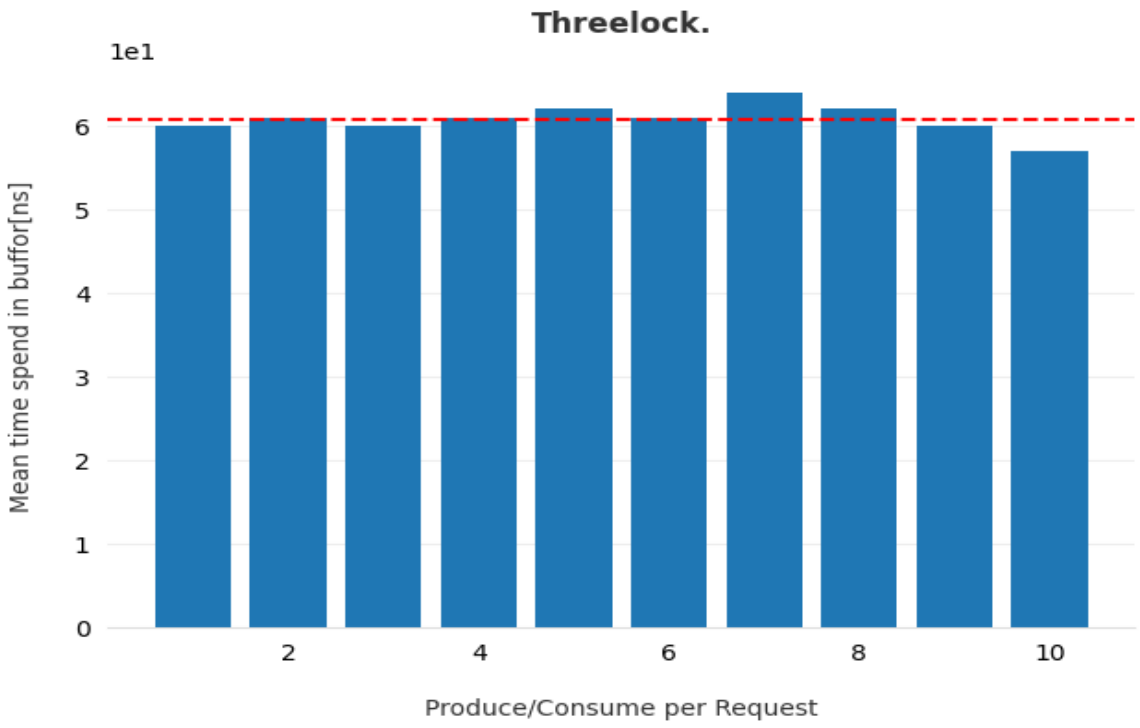
Rysunek 9: Wyniki dla ThreeLockBuffer

Wnioski
Z powyższych wykresów wynika, iż zaimplementowane bufory nie posiadają tendencji do głodzenia wątków. Poszczególne wątki spędzają podobną ilość czasu w buferze. Sytuacja ta jest szczególnie nietypowa dla bufora twoConditons. Powinien on zagładzać wątki, które są bardziej wymagające. W tym teście jednak, wątki za każdym razem losują liczbę, którą wysyłają do bufora. Nie ma więc w systemie wątków, które żądają cały czas dużej ilości elementów w buforze. W związku z tym, kolejki w buforze nie są nieustannie zapełnione wymagającymi wątkami. Wynik tego testu, sugeruje, że aby zapobiec zjawisku zagładzania w buforze twoConditions wystarczy pozbyć się wątków szczególnie „chciwych"lub wymusić na nich rzadsze wysyłanie wymagających zamówień.

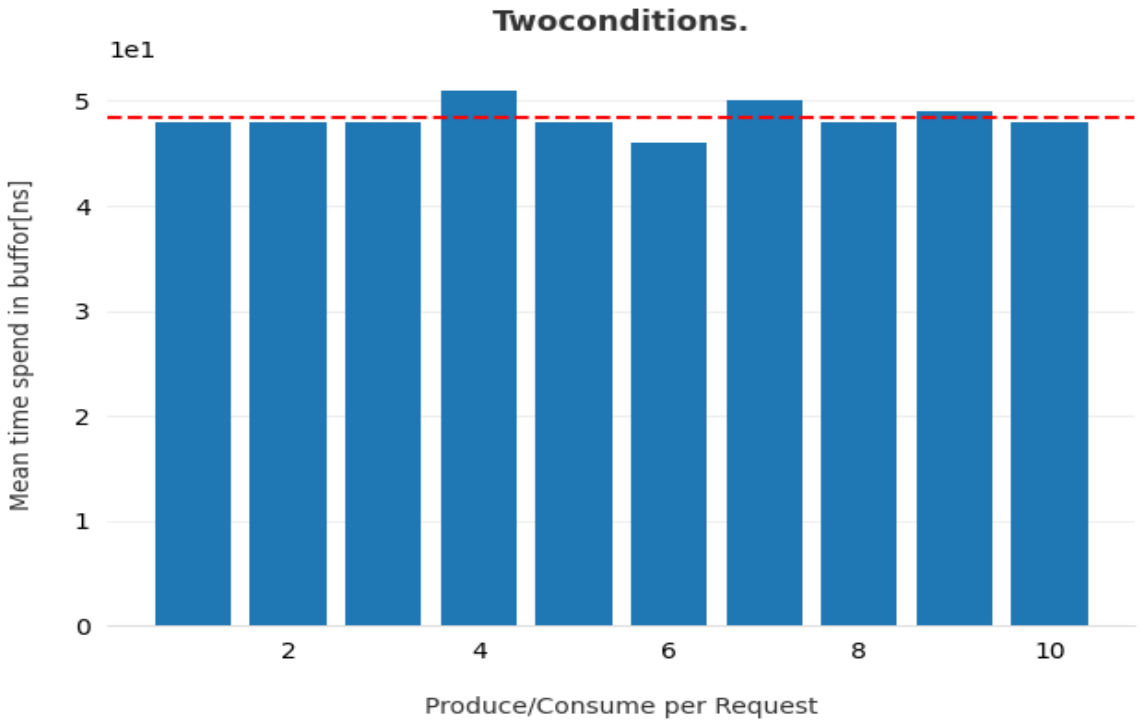
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 10



Rysunek 10: Wyniki dla FourConditionBuffer



Rysunek 11: Wyniki dla ThreeLockBuffer



Rysunek 12: Wyniki dla ThreeLockBuffer

Wnioski Zmniejszenie maksymalnej liczby wysyłanej do bufora polepszyła działanie wszystkich metod (zagładzanie). Dla każdej „prośby"wysyłanej do bufora, czas, jaki wątek spędził w monitorze przybliżył się do średniej. Zatem, jeżeli programiście zależy na nie zagładzaniu wątków, to lepiej aby odpowiednio zwiększył bufor, badź zmniejszył maksymalne zamówienie wysyłane do monitora.

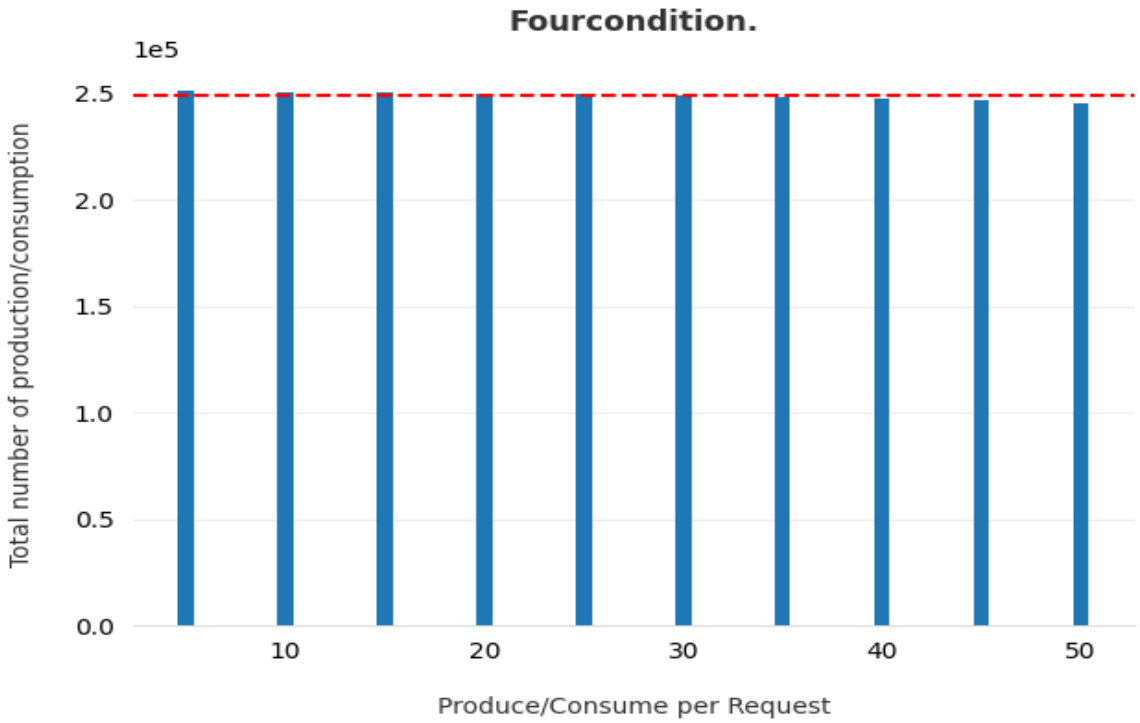
5 Wpływ metody sleep() na działanie bufore

5.1 Opis

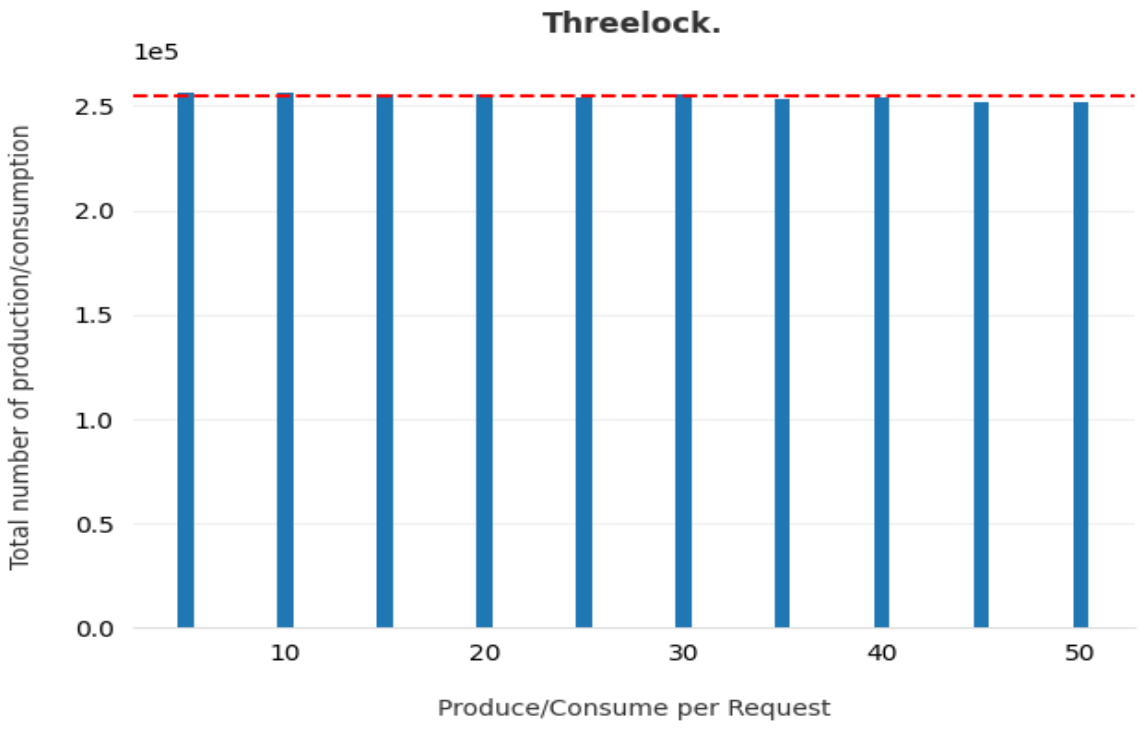
Eksperyment ten jest kopią eksperymentu numer 1. Jednak, dodatkowo, każdy wątek po wykonaniu operacji na buforze, wywołuje funkcję sleep() na jedną nanosekundę.
Parametry dla tego testu są takie same jak poprzednio.

5.2 Wyniki

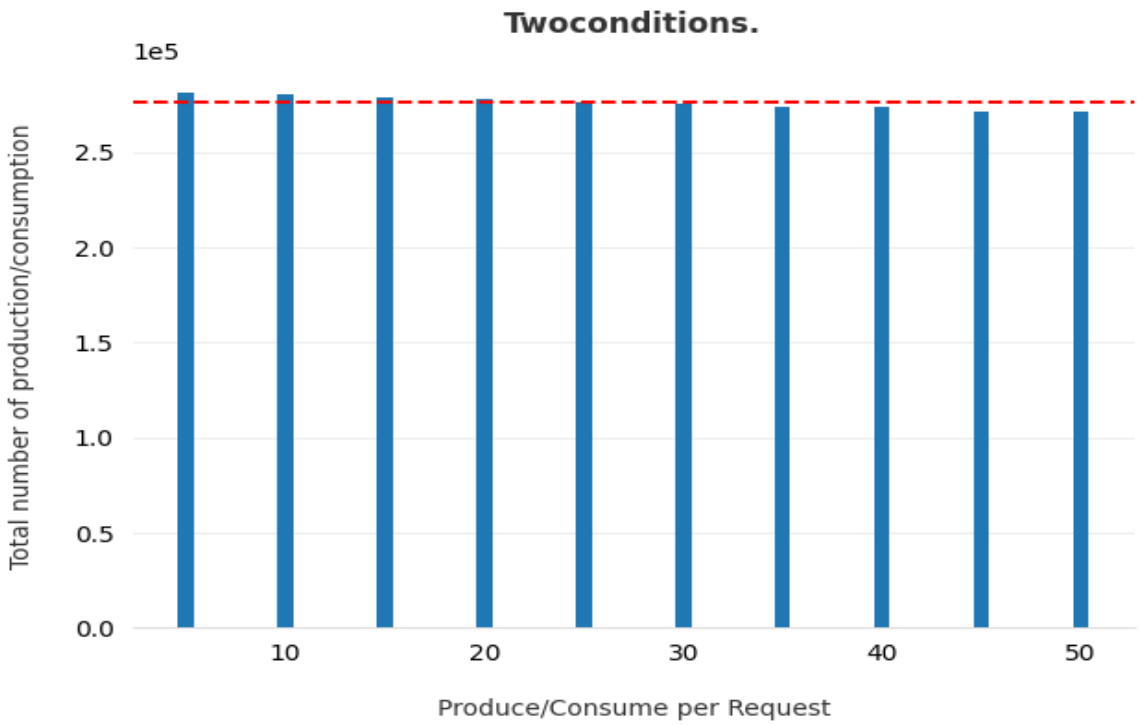
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 50



Rysunek 13: Wyniki dla FourConditionBuffer

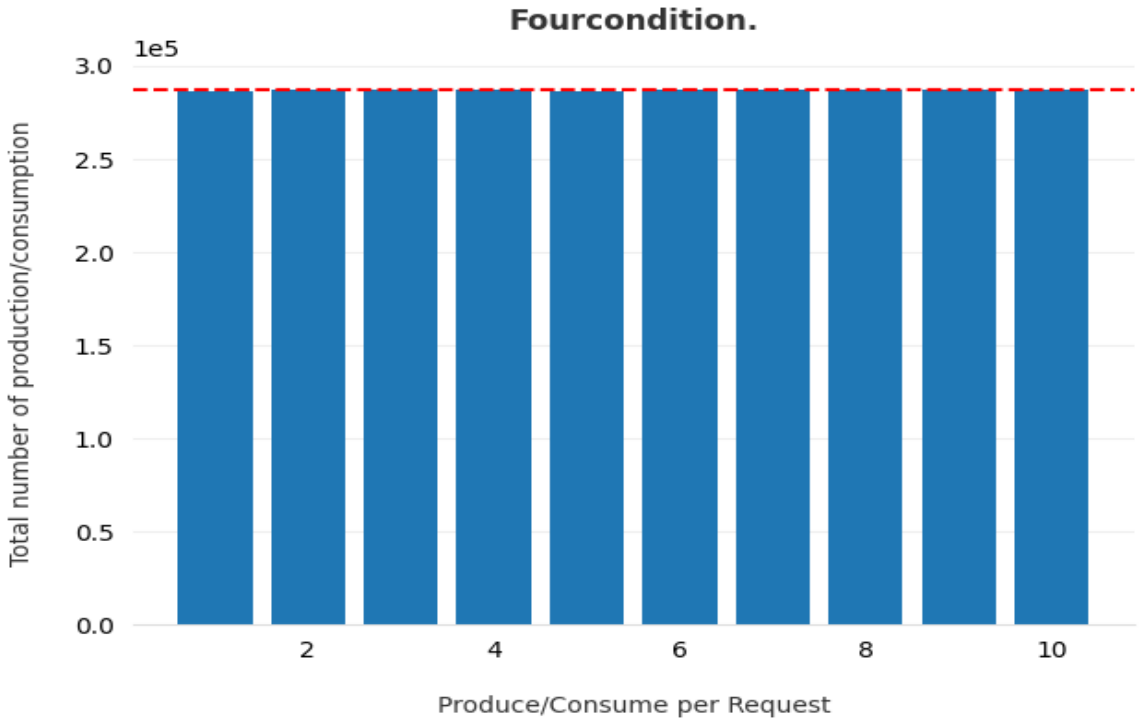


Rysunek 14: Wyniki dla ThreeLockBuffer

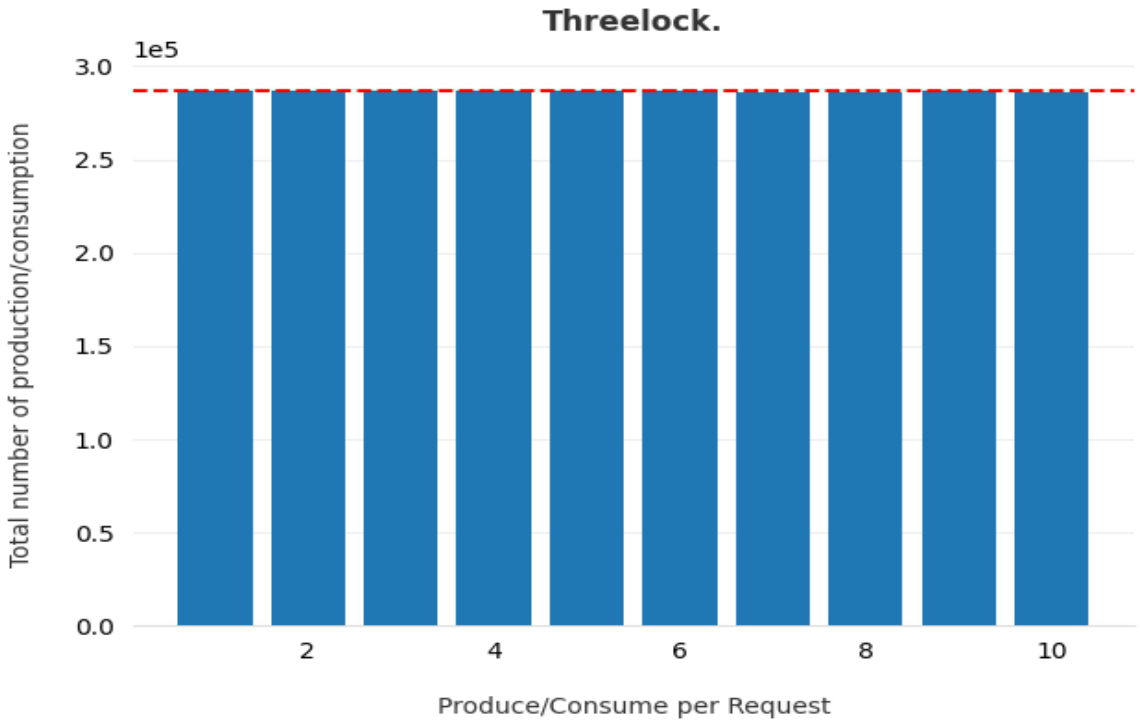


Rysunek 15: Wyniki dla ThreeLockBuffer

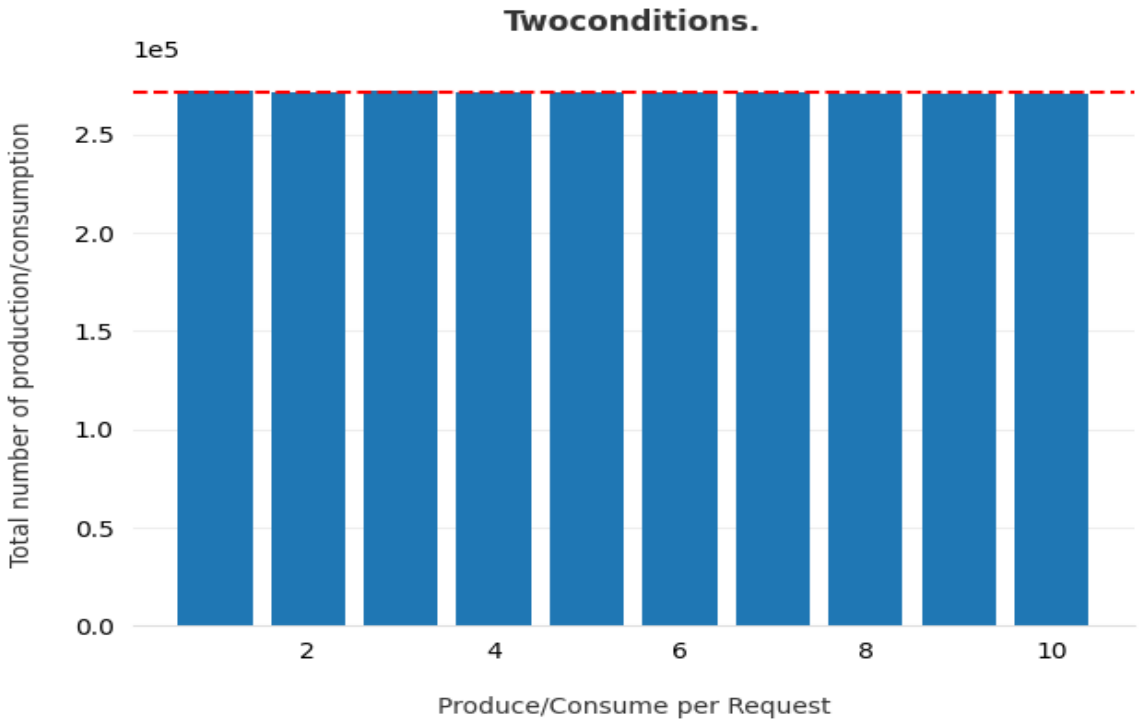
- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 10



Rysunek 16: Wyniki dla FourConditionBuffer



Rysunek 17: Wyniki dla ThreeLockBuffer



Rysunek 18: Wyniki dla ThreeLockBuffer

Wnioski W tym eksperymencie pokazano, że metoda sleep() w znaczący sposób potrafi zmienić działanie bufora. Dzięki jej zastosowaniu, zagłodzenie wątków nie występuję już w żadnej metodzie. Niestety brak zagłodzenia, występuje kosztem efektywności algorytmów. W pierwszym

eksperymencie, na osi OY widniały wartości rzędu 1e7-1e8 (dla opisanych wyżej parametrów). Tutaj maksymalna liczba operacji wykonywana na buforze jest rzędu 1e5. Zatem eliminacja zagładzania, została osiągnięta stosunkowo dużym kosztem efektywności.

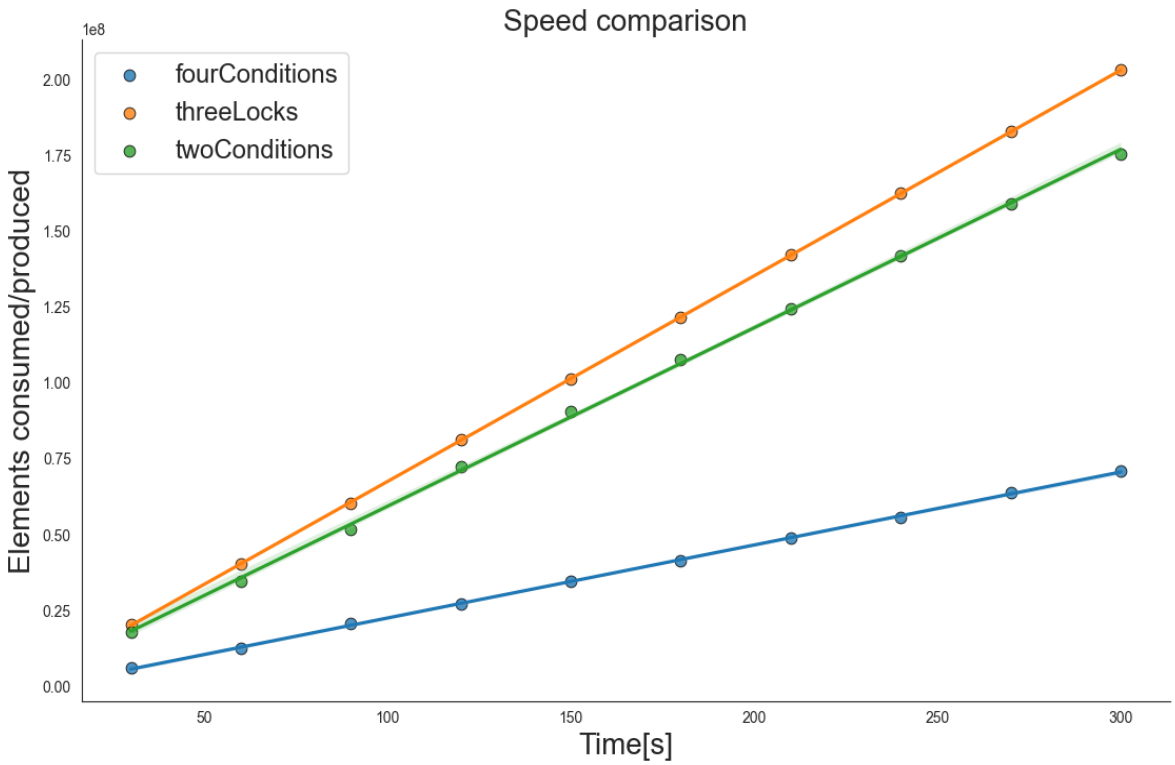
6 Pomiar efektywności

6.1 Opis

Ostatni test to pomiar efektywności zaimplementowanych metod. Każda z nich została włączona na określony czas. Cyklicznie sprawdzana była ilość operacji wykonanych na buforze (przez wszystkie wątki w systemie). Na końcu na wykresie wyświetlony została zależności operacji na buforze od czasu z prostą najlepszego dopasowania. Parametry są takie same jak dla poprzedniego eksperymentu.

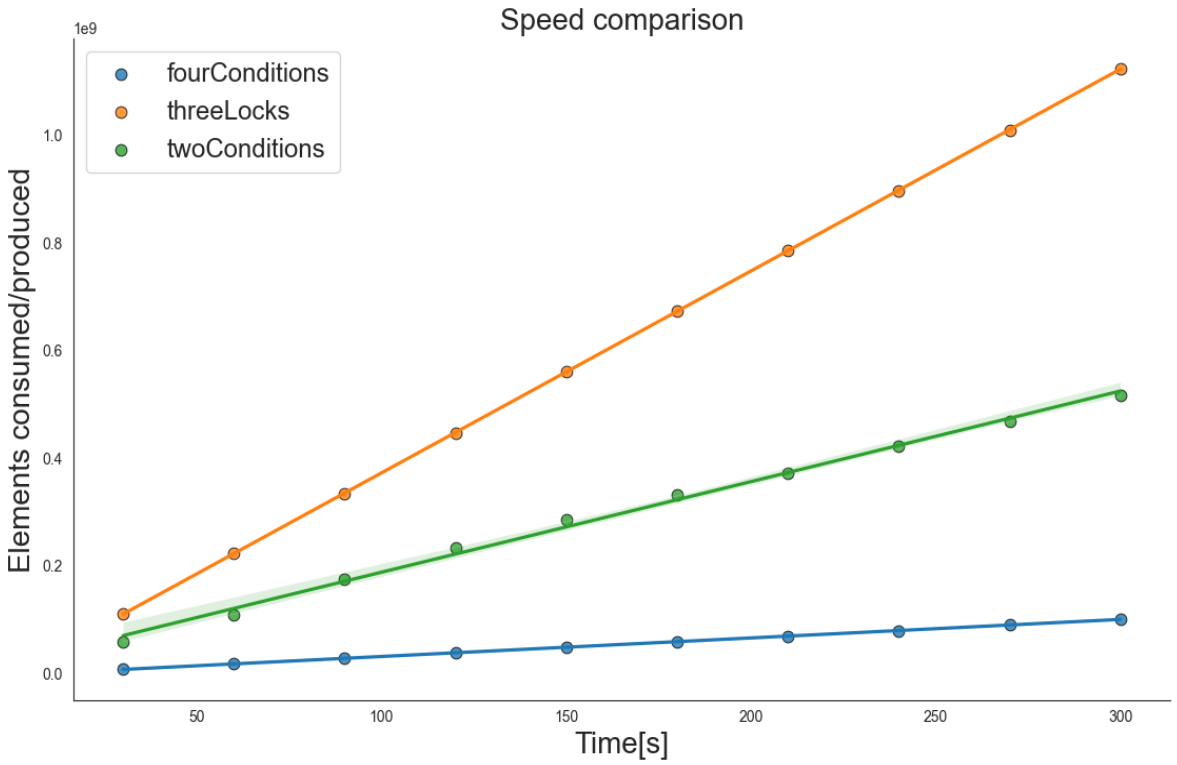
6.2 Wyniki

- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 50
  Liczba rund: 10



Rysunek 19: Porównanie efektywności

- **Parametry:** czas trwania: 5 minut
  pojemność bufora: 100
  liczba wątków klasy Producer (Consumer): 10
  Maksymalna liczba wysyłana do Bufera: 10
  Liczba rund: 10



Rysunek 20: Porównanie efektywności

7 Wnioski - podsumowanie

Analizując powyższe wykresy efektywności, można wywnioskować, że metoda oparta na trzech Lockach jest najszybsza wsród analizowanych rozwiązań. Oprócz tego bardzo dobrze poradziła sobie z problemem zagładzania. Potwierdzają to eksperymenty 1-3. Drugim, pod względem efektywności algorytmem jest twoConditions. Dla pierwszego zestawu parametrów działał porównywalnie szybko co threeLock. Niestety, wraz ze zmniejszaniem maksymalnej liczby wysyłanej do bufora, zwiększyła się również różnica pomiędzy tymi metodami. Ponadto, metoda ta posiada tendencję do zagładzania, gdy w systemie działają „chciwe"wątki. Najwolniejszą metodą okazała się być metoda FourConditions. Nie posiada ona tendencji do zagładzania wątków, ale jej efektywność jest zdecydowanie mniejsza od pozostałych rozwiązań.
Podsumowując, dla sprzętu i implementacji opisanej wyżej, najlepszym rozwiązaniem problemu Producenta-Konsumenta okazała się być metoda ThreeLocks.