Układy synchroniczne i asynchroniczne - sprawozdanie z porównania dwóch rozwiązań

Oskar Simon

Spis treści

1. Opis projektu

Zadanie polegało na rozwiązaniu problemu „Producenci i konsumenci” w sposób synchroniczny oraz asynchroniczny i przeprowadzeniu badań w celu odnalezienia dla jakich warunków które rozwiązanie jest lepsze. Rozwiązanie synchroniczne opiera się o zastosowanie trzech „lock'ów” w języku Java – jeden dla konsumentów, jeden dla producentów oraz jeden wspólny, synchronizujący dostęp do bufora. W rozwiązaniu asynchronicznym działanie programu oparte jest o wzorzec projektowy Active Object. W dalszej części sprawozdania producenci i konsumenci nazywani także będą w skrócie klientami.

2. Implementacja wzorca Active Object

Wybór struktur danych

Wzorzec Active Object zakłada w kilku kluczowych miejscach użycie odpowiednich struktur danych, dobranych do potrzeb problemu. Najbardziej interesującymi z nich są struktura typu „Future”, odpowiadająca za zwrócenie wyniku pracy servant'a (produkcji i konsumpcji) do zleceniodawcy, oraz kolejki zleceń wewnątrz scheduler'a.

Future

W rozpatrywanym przez nas problemie zleceniodawca potrzebuje jedynie odczytać wynik pracy servant'a ze struktury typu „Future”, do której pisać może jedynie servant. Z tej przyczyny jedynym negatywnym rezultatem braku synchronizacji pomiędzy wątkami jest jednorazowe odczytanie informacji o braku wykonania zadania, zamiast informacji o jego sukcesie. Jest to, w porównaniu z kosztem synchronizacji na tym etapie, zaniedbywalny problem. Z tej przyczyny klasa „Future” nie jest synchronizowana.

Kolejki scheduler'a

Kolejka zleceń w scheduler'rze z oczywistych powodów musi być synchronizowana (wiele piszących). Ponieważ jedynymi potrzebnymi operacjami w tym problemie są operacja wstawienia na koniec kolejki, oraz ściągnięcia z jej początku (te dwie operacje w większości przypadków nie muszą być synchronizowane), postanowiłem zastosować wbudowaną w bibliotekę Javy LinkedBlockingQueue.

Dodatkowo, scheduler w opisanym problemie musi obsługiwać funkcjonalność „wpuszczania” zleceń przed czekające na możliwość ich wykonania zlecenia przeciwnego rodzaju (dla produkcji konsumpcja i na odwrót). Scheduler może ustawiać czekające na możliwość ich wykonania zlecenia w kolejce, w kolejności ich ściągnięcia z kolejki synchronizowanej, a następnie ściągać je, gdy pojawi się możliwość ich wykonania. Tutaj jednak jest to funkcjonalność w pełni wewnętrzna dla wątku scheduler'a, zatem synchronizacja jest zbędna. Wybraną przeze mnie strukturą jest ArrayDeque z biblioteki Javy.

3. Opis eksperymentów

Zastosowane metryki

Głównym problemem rozpatrywanym w tym projekcie jest różnica w wydajności obu rozwiązań. Z tego powodu postanowiłem zająć się pomiarem wykonanej przez wątki pracy w zadanym czasie. Pracę tę zdefiniowałem poprzez liczbę wykonanych dodatkowych zadań w wątkach producentów/konsumentów oraz w wątku servant'a.

Metodologia przeprowadzania pomiarów - parametryzacja

Pomiary wykonywane były dla następujących parametrów:

1) **Pojemność bufora i maksymalna wielkość produktu**

   W pierwszej części badań wielkość bufora została ustawiona na 2.0e4, jego początkowa wartość na 1.0e4, a maksymalna wielkość produktu na 5. Miało to na celu stworzenie warunków, w których wątki mogą bez problemu wykonywać produkcję oraz konsumpcję.

2) **Liczba producentów i liczba konsumentów**

   W celu stworzenia warunków, w których zlecenia przychodzą w relatywnie „płynny" sposób, ale Java nie zaczyna mieć przesytu w liczbie wątków, ich liczba została trwale ustawiona na 20 (zarówno producentów jak i konsumentów).

3) **Czas wykonywania programu**

   Trwale ustawiony na 10^4 ms (10 sekund). Po tym czasie wątki sumują liczbę zadań, które zdążyły wykonać i kończą działanie.

4) **Wagi dodatkowej pracy w producencie/konsumencie oraz pracy servant'a**

   Dodatkowa praca polegała na zasypianiu określonego wątku na zadany czas. Programy wykonywane były dla wszystkich kombinacji długości snu producenta/konsumenta oraz servant'a z przedziału [10ms, 100ms] ze skokiem 10ms (łącznie 100 wykonań). Dla szerszego obrazu wykonałem także pomiary dla przedziału [100ms, 1000ms] ze skokiem 100ms (łącznie także 100 wykonań).

   Dodatkowo, w następnej części eksperymentu przeprowadziłem badania dla „zerowej" pracy dodatkowej oraz dla „zerowej" pracy servant'a.

Środowisko testowe

System Windows 10 x64

Procesor i5-10300H

Pamięć RAM 16GB

Środowisko IntelliJ 2021.2.2

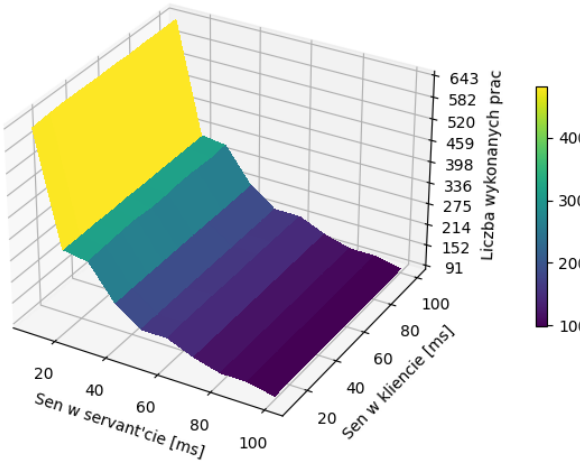
Język programowania Java 18.0

4. Wyniki testów

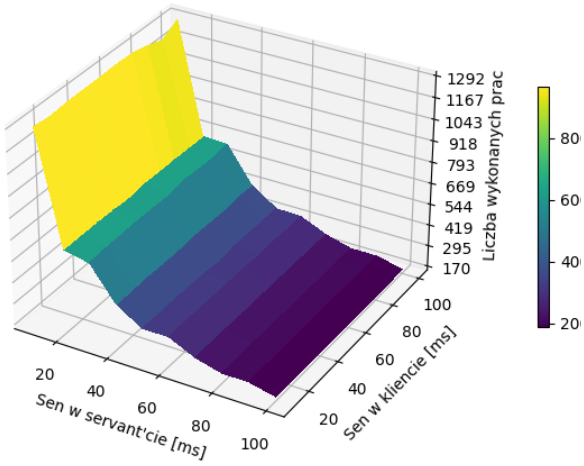
Praca w zakresie [10ms, 100ms]

Praca na buforze

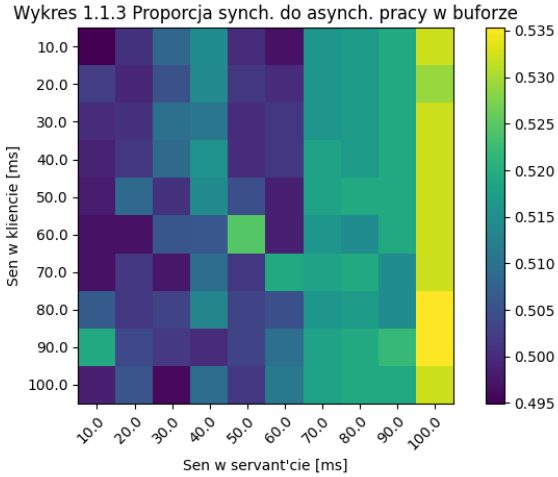
Wykres 1.1.1. Synchroniczna praca na buforze



Wykres 1.1.2. Asynchroniczna praca na buforze



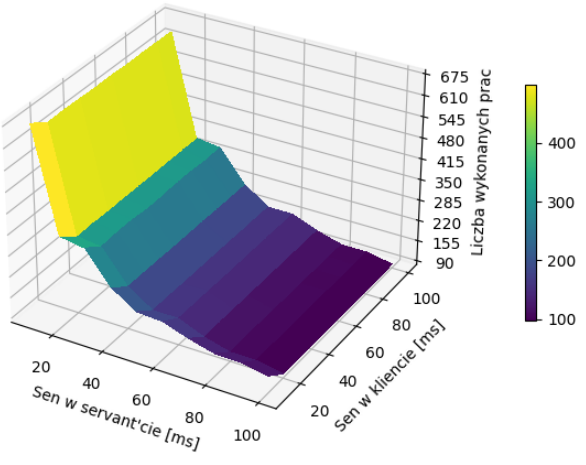
Wykres 1.1.3 Proporcja synch. do asynch. pracy w buforze



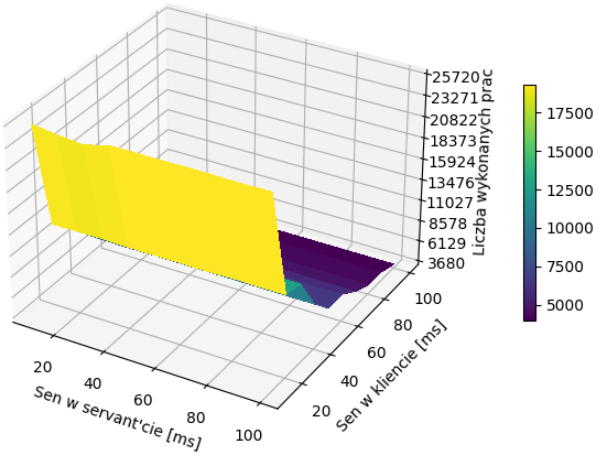
Wykresy 1.1.1. oraz 1.1.2. pokazują, jak niewielki wpływ na całkowitą pracę na buforze ma praca wewnątrz klienta. Jest to spowodowane faktem, że dostarczyliśmy buforowi wystarczającą liczbę klientów, by zawsze był zajęty. Wykres 1.1.3 natomiast wskazuje, że system servant'a w rozwiązaniu synchronicznym jest w znacznie bardziej zajęty kolejkowaniem klientów, przez co spędza mniej czasu na docelowej pracy.

Dodatkowa praca klienta

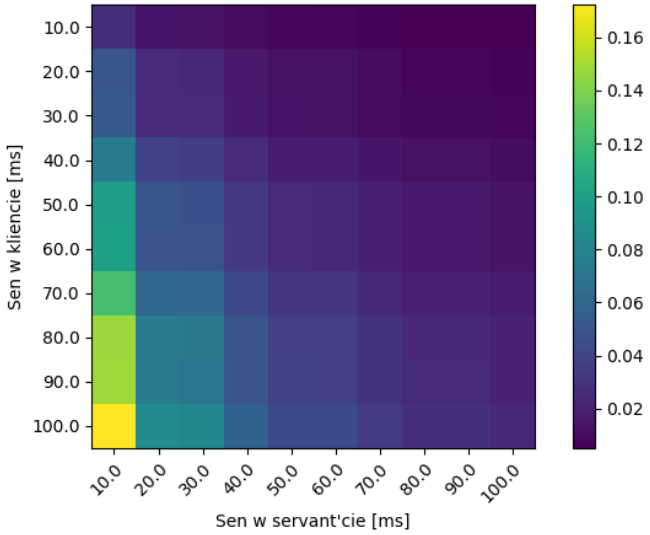
Wykres 1.2.1. Synchroniczna praca w kliencie



Wykres 1.2.2. Asynchroniczna praca w kliencie



Wykres 1.2.3 Proporcja synch. do asynch. pracy klienta



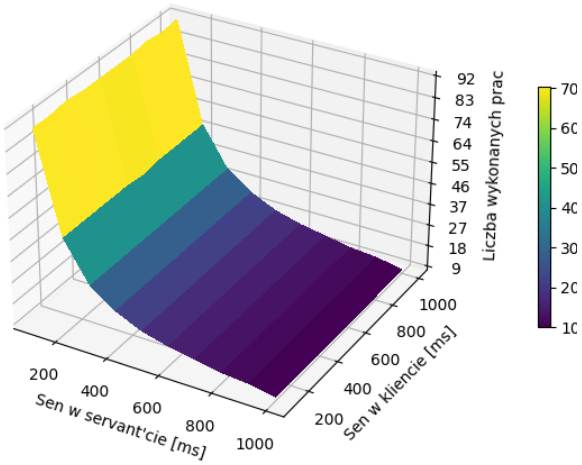
Wykresy 1.2.1. oraz 1.2.2 wskazują znaczącą różnicę w zachowaniach obu rozwiązań. W rozwiązaniu synchronicznym, w dodatkowej pracy klientów, długość snu klienta nie robiła dużej różnicy. Natomiast w rozwiązaniu asynchronicznym jest na odwrót. Podczas gdy długość pracy wewnątrz servant'a nie miała znaczącego wpływu na liczbę wykonanych przez klientów dodatkowych prac, ich długość już tak.

Na wykresie 1.2.3 widać, że rozwiązanie asynchroniczne znacznie bardziej dba o dodatkową pracę klientów. Zgadza się to z głównymi założeniami, więc implementacja jest najprawdopodobniej poprawna.

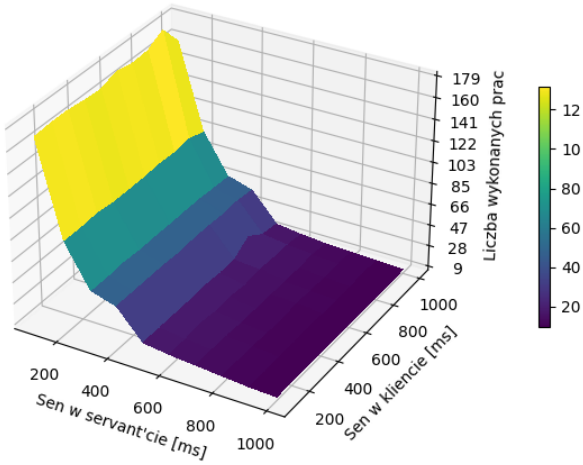
Podatkowa praca w zakresie [100ms, 1000ms]

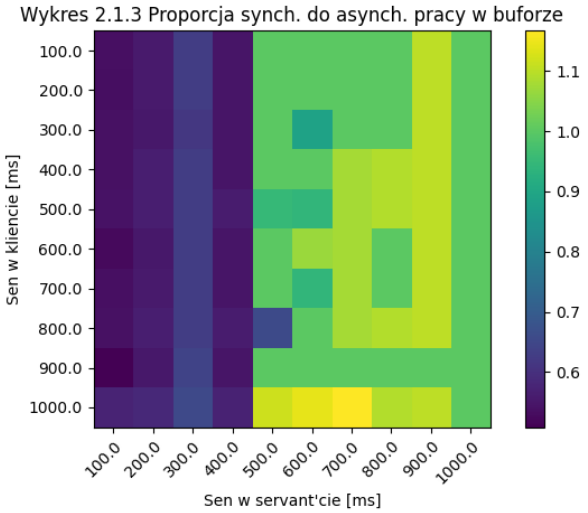
Praca na buforze

Wykres 2.1.1. Synchroniczna praca w buforze



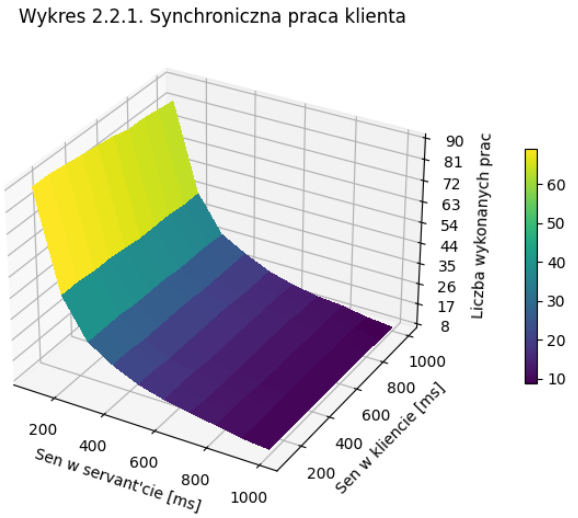
Wykres 2.1.2. Asynchroniczna praca w buforze

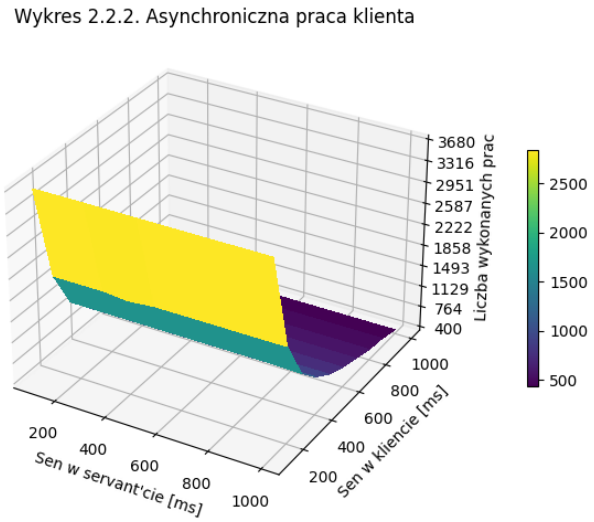


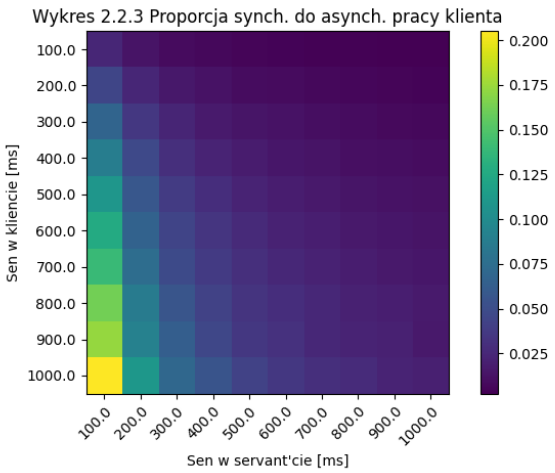


Choć wykresy 2.1.1. oraz 2.1.2. nie wnoszą dużo więcej niż 1.1.1. oraz 1.1.2., na wykresie 2.1.3 widać znaczną poprawę rozwiązania synchronicznego. Okazuje się, że istnieją odpowiednie parametry zadania, dla których rozwiązanie synchroniczne jest lepsze od asynchronicznego, pod względem pracy wykonanej w buforze.

Dodatkowa praca klienta



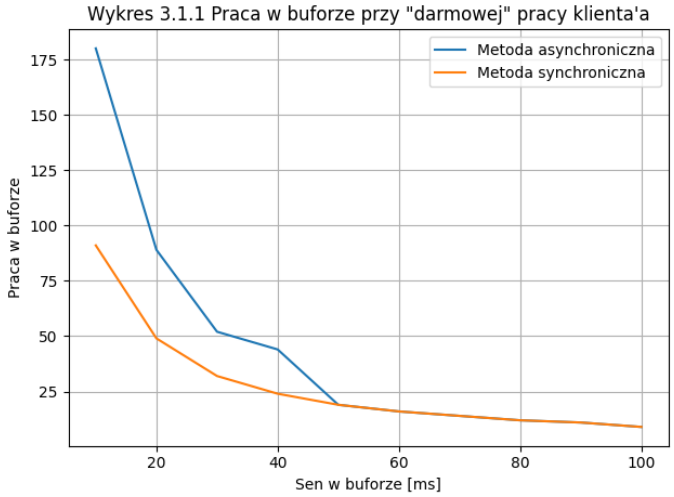


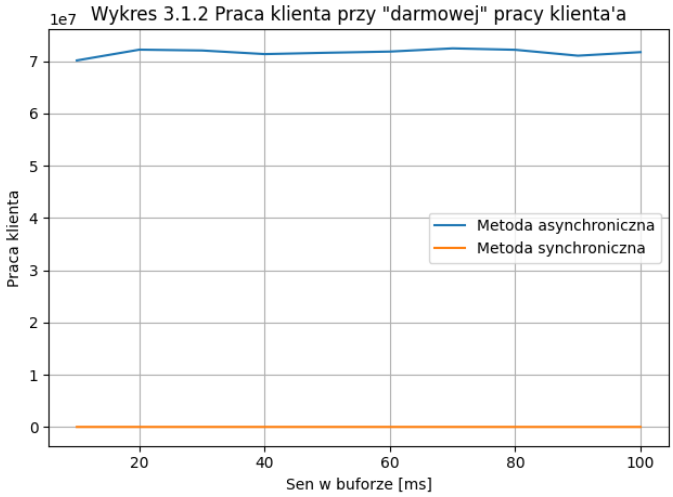


Podobnie jak ostatnio, wykresy 2.2.1. oraz 2.2.2. pokazują niewiele więcej od wykresów 1.2.1. oraz 1.2.2.. Wykres 2.2.3 natomiast pokazuje, że choć rozwiązanie synchroniczne wychodzi na prowadzenie względem asynchronicznego pod względem pracy na buforze, dodatkowa praca klienta w większości przypadków nadal wykonywana jest znacznie mniejszą liczbę razy.

Praca „zerowa”

Brak dodatkowej pracy producenta/konsumenta

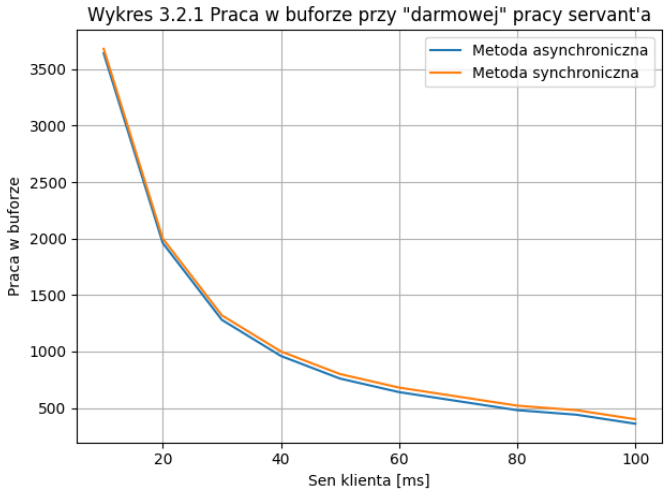


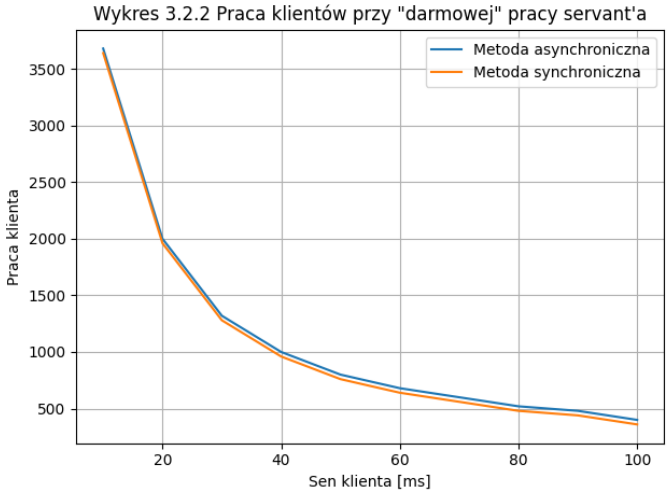


Wykres 3.1.1 wskazuje, jak dla niskich długości pracy w buforze rozwiązanie asynchroniczne nadal pozostaje lepsze od rozwiązania synchronicznego, nawet przy „zerowej” pracy klienta.

Wykres 3.1.2 natomiast może być traktowany bardziej jako ciekawostka i potwierdzenie, że programy działają tak, jak powinny. Widać na nim jedynie różnice w częstości odwiedzania pętli w wątku klienta w obu rozwiązaniach.

Brak dodatkowej pracy servant’a





Wykresy 3.2.1 oraz 3.2.2 pokazują jak oba rozwiązania „faworyzują” pewien rodzaj pracy nad inny, w momencie gdy praca w buforze jest w przybliżeniu darmowa.

5. Wnioski

W przytłaczającej większości przypadków rozwiązanie asynchroniczne daje lepsze wyniki zarówno pod względem pracy wykonanej na buforze, jak i pracy dodatkowej, wykonywanej przez producentów/konsumentów. Istnieją jednak przypadki, w których rozwiązanie synchroniczne staje się istotnie lepsze (rząd kilkunastu procent wydajności) pod względem pracy wykonywanej na buforze. Staje się to w momencie, gdy praca w buforze staje się odpowiednio długa, prawdopodobnie przez fakt, że czas poświęcony na kolejkowanie klientów staje się pod względem tej pracy znikomy. Mimo to, osoba wybierająca to rozwiązanie powinna mieć na uwadze fakt, że dodatkowa praca wykonywana przez klientów jest wtedy znacznie mniejsza, niż w rozwiązaniu asynchronicznym.

Próbując założyć luźnie sformułowaną ogólną regułę, można przyjąć, że do większości przypadków bezpiecznie jest stosować rozwiązanie asynchroniczne, ponieważ zazwyczaj daje najlepszą wydajność. Gdy jednak skupiamy się na pracy na buforze, a dodatkowa praca klientów jest dla nas zadaniem drugoplanowym, rozwiązanie synchroniczne zazwyczaj (choć niestety także nie zawsze) jest lepsze.