

Programowanie rozproszone

Jakub Kwiatkowski 145356

Paweł Strzelczyk 145217

Problem

W winiarzy (oznaczonych dalej przez W_i) produkuje każdy po X_i litrów wina.

S studentów (oznaczonych dalej przez S_j) konsumuje każdy po Y_j litrów wina.

Aby przekazać wino, winiarz musi wynająć bezpieczne miejsce B_k .

Winiarz nie rozpocznie produkcji wina, dopóki nie odda wszystkiego, co już wyprodukował.

Przy założeniach:

$$i \in \{1, 2, \dots, W\}$$

$$j \in \{1, 2, \dots, S\}$$

$$k \in \{1, 2, \dots, N\}$$

$$\neg(\forall_{i \in \{1, 2, \dots, W\}} \exists_{j \in \{1, 2, \dots, S\}} X_i = Y_j)$$

Proponowane rozwiązanie

Aby rozwiązać podany problem musimy poczynić dodatkowe założenia:

- $\sum X_i = \sum Y_j$
- Winiarz nie musi oddać całej partii jednemu studentowi ($X_i \geq Y_j$)
- Student nie musi zaspokoić wszystkich swoich potrzeb u jednego winiarza ($Y_j \geq X_i$)
- Student może w razie potrzeby zaspokoić tylko część swojego zapotrzebowania, jednak pozostała część musi zostać zaspokojona tak szybko jak to tylko możliwe.

Założenie pierwsze zapobiega problemowi nadprodukcji. Jeśli założymy że proces jest ciągły i nieskończony, to przybiera ono formę:

$$\lim_{n \rightarrow \infty} n \times \sum X_i = \lim_{n \rightarrow \infty} n \times \sum Y_j$$

i jest pomijalne w praktyce.

Założenia drugie i trzecie zapobiegają sytuacji, w której popyt i podaż sumarycznie się równoważą, ale niemożliwy jest przydział całościowy.

Założenie czwarte jest potrzebne w przypadku procesu ciągłego i nieskończonego, pozwalając na złamanie założenia pierwszego w czasie jednej iteracji zakładając, że zostanie ono skorygowane w czasie następnych iteracji (zbyt mała podaż w i -tej iteracji zostanie zrównoważona nadpodażą w $i + 1$ iteracji)

Opis algorytmu

1. Winiarz W_i „produkuje” X_i litrów wina i ubiega się o bezpieczne miejsce B_i :
 - (a) W_i rozsyła do wszystkich winiarzy wiadomość **REQ** zawierającą swój zegar Lamporta L_{W_i} oraz żądane miejsce B_i .
 - (b) Po otrzymaniu wiadomości **ACK** (lub **REQ** z wyższymi zegarami Lamporta niż własny) od wszystkich winiarzy, W_i wchodzi do sekcji krytycznej i rozsyła wszystkim wiadomość M_{W_i} (**INFO**) zawierającą informacje o B_i i X_i .
 - (c) Jeśli W_i otrzyma wiadomość **REQ** z zegarem Lamporta niższym niż własny odsyła **ACK** i dalej oczekuje na pozostałe **ACK**.
 - (d) Po rozgłoszeniu wiadomości M_{W_i} , W_i wychodzi z sekcji krytycznej i rozsyła **ACK** wszystkim oczekującym.
2. W_i rozpoczyna następną produkcję dopiero po otrzymaniu **RELEASE** zawierającego B_i .
3. Student S_j określa swoje zapotrzebowanie na Y_j litrów wina i ubiega się o dostęp do B_k :
 - (a) S_j wysyła do wszystkich studentów wiadomość **REQ** zawierającą B_k oraz ilość wina jaką zamierza stamtąd pobrać.
 - (b) Po otrzymaniu wiadomości **ACK** (lub **REQ** z wyższymi zegarami Lamporta niż własny) od wszystkich studentów, S_j wchodzi do sekcji krytycznej.
 - (c) Jeśli S_j wyczerpie zasoby B_k , to rozsyła winiarzom wiadomość **RELEASE** zawierającą B_k .
 - (d) Po wyjściu z sekcji krytycznej S_j rozsyła **ACK** wszystkim oczekującym.
4. S_j ubiega się o kolejne B_k doputy, dopóki nie wypełni zapotrzebowania Y_j .

Zegary Lamporta są inkrementowane w zwykły sposób, jednak nigdy nie w trakcie rozsyłania wiadomości (rozesłanie wiadomości do grupy odbiorców jest traktowane jako operacja atomowa).

Implementacja

W naszej implementacji proponowanego algorytmu przyjęliśmy następującą strukturę wiadomości:

```
Message {  
    type,  
    timestamp,  
    sender,  
    payload {  
        safehouse_index,  
        wine_volume,  
        last_timestamp  
    }  
}
```

Pole **payload** jest zawartością specyficzną dla typu wiadomości. Poniżej przedstawiono typy wiadomości i pole **payload** specyficzne dla nich.

- WINEMAKER_REQUEST:

```
payload: {
    safehouse_index
}
```

- WINEMAKER_ACKNOWLEDGE:

```
payload: {}
```

- WINEMAKER_BROADCAST:

```
payload: {
    safehouse_index,
    wine_volume
}
```

- STUDENT_REQUEST:

```
payload: {
    safehouse_index,
    wine_volume
}
```

- STUDENT_ACKNOWLEDGE:

```
payload: {
    safehouse_index,
    last_timestamp
}
```

- STUDENT_BROADCAST:

```
payload: {
    safehouse_index
}
```

Niech \mathbb{W} będzie uporządkowanym wektorem winiarzy oraz \mathbb{S} będzie uporządkowanym wektorem studentów. Ponadto, niech \mathbb{B} będzie wektorem bezpiecznych miejsc. Wtedy:

$\forall w \in \mathbb{W}$:

1. Niech i będzie indeksem w w wektorze \mathbb{W}
2. Niech $b = i \bmod \|\mathbb{B}\|$ będzie przypisanym do w indeksem bezpiecznego miejsca w wektorze \mathbb{B}
3. Niech c będzie zegarem lamporta przypisanym do w .

4. w zwiększa c o 1 i rozsyła do pozostałych winiarzy wiadomość `WINEMAKER_REQUEST` z informacją o b
5. w ustawia swój licznik zgód na 0
6. Dopóki licznik zgód jest mniejszy od $\|\mathbb{W}\| - 1$:
 - (a) w oczekuje na wiadomość m
 - (b) w ustawia $c = \max\{c, m.timestamp\}$
 - (c) Jeśli m jest typu `WINEMAKER_ACKNOWLEDGE`, w zwiększa licznik zgód o 1
 - (d) Jeśli m jest typu `WINEMAKER_REQUEST`:
 - i. Jeśli zegar m jest niższy od zegara wysłanej wiadomości lub indeks bezpiecznego miejsca jest różny od b , w wysyła wiadomość zwrotną `WINEMAKER_ACKNOWLEDGE`
 - ii. Jeśli zegar m jest równy zegarowi wysłanej wiadomości oraz identyfikator w jest większy od identyfikatora nadawcy m , w wysyła wiadomość zwrotną `WINEMAKER_ACKNOWLEDGE`
 - iii. Jeśli zegar m jest większy od zegara wysłanej wiadomości lub zegar m jest równy zegarowi wysłanej wiadomości oraz identyfikator w jest mniejszy od identyfikatora nadawcy m , w dopisuje nadawcę m do kolejki oczekujących na zgodę i zwiększa licznik zgód o 1
7. w wybiera losowo liczbę jednostek wina v , i rozsyła ją do wszystkich studentów wiadomością `WINEMAKER_BROADCAST`
8. w oczekuje na wiadomość m typu `STUDENT_BROADCAST` z indeksem bezpiecznego miejsca równym b
9. w zwiększa c o 1 i rozsyła wiadomość `WINEMAKER_ACKNOWLEDGE` do winiarzy w kolejce oczekujących na zgodę
10. Wróć do 4

$\forall s \in \mathbb{S}$:

1. Niech $b \in \mathbb{B}$ będzie bezpiecznym miejscem
2. Niech S_b będzie liczbą jednostek składowanych w b
3. Niech c będzie zegarem lamporta przypisanym do s
4. s wybiera losowo liczbę jednostek wina v
5. s wybiera pierwsze niepuste bezpieczne miejsce b
 - (a) Jeśli wszystkie b są puste, s oczekuje na wiadomość m :
 - (b) s ustawia $c = \max\{c, m.timestamp\}$
 - (c) Jeśli m jest typu `WINEMAKER_BROADCAST`, s aktualizuje stan odpowiedniego b , oraz wybiera b jako swoje bezpieczne miejsce

- (d) Jeśli m jest typu `STUDENT_REQUEST`, s zwiększa c o 1 i wysyła wiadomość zwrotną `STUDENT_ACKNOWLEDGE`
- 6. s zwiększa c o 1 i rozsyła do wszystkich studentów wiadomość `STUDENT_REQUEST` z informacją o b i v , oraz ustawia znacznik p (priorytet) na c .
- 7. s ustawia swój licznik zgód na 0
- 8. Dopóki licznik zgód jest mniejszy od $\|\mathbb{S}\| - 1$:
 - (a) s oczekuje na wiadomość m
 - (b) s ustawia $c = \max\{c, m.timestamp\}$
 - (c) Jeśli m jest typu `STUDENT_ACKNOWLEDGE` oraz $m.payload.last_timestamp$ jest równe p , s zwiększa licznik zgód o 1
 - (d) Jeśli m jest typu `STUDENT_REQUEST`:
 - i. Jeśli zegar m jest niższy od p lub indeks bezpiecznego miejsca jest różny od b , s zwiększa c o 1 i wysyła wiadomość zwrotną `STUDENT_ACKNOWLEDGE` oraz zmniejsza odpowiednio zapasy w bezpiecznym miejscu wskazanym w m
 - ii. Jeśli zegar m jest równy p oraz identyfikator s jest większy od identyfikatora nadawcy m , s zwiększa c o 1 i wysyła wiadomość zwrotną `STUDENT_ACKNOWLEDGE` oraz zmniejsza odpowiednio zapasy w bezpiecznym miejscu wskazanym w m
 - iii. Jeśli zaszła jedna z powyższych sytuacji i $S_b = 0$, s porzuca próbę zajęcia b i wraca do punktu 5
 - iv. Jeśli zegar m jest większy od zegara wysłanej wiadomości lub zegar m jest równy zegarowi wysłanej wiadomości oraz identyfikator s jest mniejszy od identyfikatora nadawcy m , s dopisuje nadawcę m do kolejki oczekujących na zgodę i zwiększa licznik zgód o 1
 - (e) Jeśli m jest typu `WINEMAKER_BROADCAST`, s aktualizuje stan odpowiedniego b
- 9. s zwiększa c o 1.
- 10. s zmniejsza S_b oraz v o $\min\{S_b, v\}$
- 11. Jeśli $S_b = 0$, s zwiększa c o 1 i rozsyła wszystkim winiarzom wiadomość `STUDENT_BROADCAST` z informacją o b
- 12. s zwiększa c o 1 i rozyła wiadomość `STUDENT_ACKNOWLEDGE` do studentów w kolejce oczekujących na zgodę
- 13. Jeśli $v \neq 0$, s wraca do punktu 5
- 14. W przeciwnym przypadku s wraca do punktu 4