Przetwarzanie rozproszone - opis algorytmu

Krystian Birecki 136683, Adam Chudzyński 136692

STRUKTURY

int N - ilość procesów

int request_num = ilość requestów stworzonych przez burmistrza

int my_clock = wartość zegara lamporta procesu.

Z - liczba zleceń

A - liczba agrafek

T - ilość trucizny

tablice: zlecenia[Z],

request my_request; - aktualne zlecenie skrzata

```
struct request{
int id,
int hamster
}

struct message{
        int type; //typ wiadomości np KILL_REQUEST
        int lamport_clock; //wartość zegara lamporta
        int sender_id, //id wysyłającego
        request requests[];
        int a; liczba oznaczająca ilość agrafek
        int poison; liczba oznaczająca ilość trucizny
        int num_rq; // wybrany numer requesta
}
```

ALGORYTM

1. INIT: Burmistrz daje sygnał broadcastowy KILL_REQUEST = 1 o tym, że utworzono request_num zleceń, w wiadomości znajduje się stan początkowy zasobu agrafek, trucizny oraz sama tablica zleceń.

```
//burmistrz
generate_requests();
```

2. Skrzaty odbierają sygnał 1 o tym, że pojawiły się nowe zlecenia oraz zapisują lokalnie dane o agrafkach, truciźnie oraz dostępnej liście zleceń Skrzaty zaczynają ubiegać się o dostęp do sekcji krytycznej zleceń z wykorzystaniem zmodyfikowanego algorytmu Ricarta Agrawali. Każdy skrzat losuje z listy dostępnych zleceń zlecenie, które chciałby wykonać, a następnie wysyła pozostałym skrzatom wiadomość o typie WANT_REQUEST = 2, w wiadomości zawiera id zlecenia, które chce wykonać. Jeżeli skrzat otrzyma N-1 odpowiedzi REQUEST_ACCEPT = 3 z nr swojego zlecenia, może wejść do sekcji krytycznej. Jeśli otrzyma chociaż jedną wiadomość REQUEST_REJECT = 4 ze swoim numerem zadania losuje inne zadanie z listy pozostałych zadań. Wszystkie wiadomości REQUEST_REJECT zawierają zadanie. Skrzaty usuwają te zadania z tablicy lokalnych zadań, ponieważ ta wiadomość oznacza, ze nie będą mogły już wziąć danego zlecenia. Skrzaty wysyłają odpowiedź request_accept tylko wtedy, gdy:

    - nie konkurują o dane zlecenie
    - mają znacznik czasu wyższy niż zlecenie (bądź w przypadku równych znaczników mają wyższe id)

    Procesy odsyłają REQUEST_REJECT gdy:

    - konkuruja o to samo zlecenie i mają znacznik czasu niższy niż wysyłający
    - są w trakcie wykonywania danego zlecenia bądź już je wykonały

    ```
    // skrzat

    request my_request = random(requests);

    message.sender_id = my_id
    message.type = WANT_REQUEST
    message.lamport_clock = my_clock
    message.num_rq = my_request.id

    send_to_all(message) // burmistrz te wiadmość ignoruje

    y_req = 0 // licznik do zliczania odpowiedzi pozytywnych na nasz sygnał
    temp_lamport = my_clock

    WANT_REQUEST
            while(y_req!= N-1)
    {
    ```

```
        recv(message)
        temp_lamport = max(message.lamport_clock,temp_lamport)+1;
        if(message.type == REQUEST_ACCEPT && message.num_rq ==
my_request.id)
        { y_req++}
        if(message.type == REQUEST_REJECT && message.num_rq ==
my_request.id){
        requests.remove(my_request)
        my_clock = temp_lamport
break;
//wracamy do początku i losujemy kolejne zadanie jeśli zadań brak czekamy
ponownie na sygnał od burmistrza
}
        if(message.type == REQUEST_REJECT){
        requests.remove(message.num_rq);

}
        if(message.type == WANT_REQUEST && message.num_rq == my_request{
        if(message.lamport_clock> my_clock || (message.lamport_clock == my_clock
&& my_id > message.sender_id){
send(message.sender_id, REQUEST_ACCEPT)}
else{ send(message.sender_id, REQUEST_REJECT)}
}
}

//Jeżeli udało się utrzymać zlecenie i opuścić pętlę można zacząć konkurować o
sprzęt

my_clock = temp_lamport;
```

W wypadku tego algorytmu w najlepszym wypadku każdy skrzat trafi na inne zlecenie i wyśle zaledwie jedną wiadomość i wiadomości potwierdzające dla innych procesów w najgorszym wypadku wszystkie procesy za każdym razem będą wybierały jedno zlecenie co doprowadzi nam do złożoności liniowej N(liczba procesów)

4. Skrzaty konkurują ponownie o agrafki. Tym razem również posłużymy się algorytmem Ricarta Agrawali ale tym razem w trochę innej formie. Skrzaty wysyłają do siebie wiadomość z sygnałem WANT_A = 5. Następnie odbierają wiadomości OK_A = 6. Jeśli odbiorą takich wiadomości N - 1 - A(liczba agrafek) oznacza to, że w mogą wziąć agrafkę i wejść do kolejnej sekcji krytycznej. Skrzaty wysyłają sygnał OK_A tylko wtedy gdy:

- mają znacznik czasowy wyższy niż wysyłający lub w przypadku równych znaczników mają wyższe id

- zakończyły wykonywanie zlecenia tym samym zwalniając agrafkę
- nie konkurują o daną sekcję krytyczną(tzn. że np skrzaty oczekujące na nowe zlecenia też dają OK_A);

```
//skrzat
<List> not_give_a ; //lista zawierające id, którym nie wysłalismy OK_A, po wykonaniu zadania skrzat powinien wysłać id z listy wiadomość OK_A

        given_a = 0; // licznik zliczający ile procesów odesłało nam wiadomość OK_A

        message.type = WANT_A
        message.lamport_clock = my_clock;
        message.sender_id = my_id;
        send_to_all(message)
        temp_lamport = my_clock;
        while(given_a!= N-1-A){
        recv(message)
        temp_lamport = max(temp_lamport, message.lamport_clock) +1;
        if( message.type = OK_A) given_a++
        if( message.type = WANT_A){
        if(message.lamport_clock < my_clock or (message.lamport_clock ==
my_clock and message.sender_id < my_id)) send(message.sender_id, OK_A);
        else {not_give_a.append(message.sender_id);}
}

}
my_clock = temp_lamport;
```

Złożonośc tego algorytmu to N. Jedyną różnicą pomiędzy oryginalnym podejściem jest to, że oryginalnie czekamy na odpowiedź(zgodę) od wszystkich ubiegających się. Natomiast w naszym przypadku wiemy, że do sekcji krytycznej agrafek może być jednoczesnie A procesów, została wprowadzona modyfikacja.
Po wyjściu z etapu oczekiwania skrzat może pobrać agrafke i rywalizować o potrzebną mu truciznę.

3. Do konkurowania o truciznę równiez wykorzystamy zmodyfikowany algorytm Ricarta, jednak tym razem będziemy musieli wziąć pod uwagę ilość potrzebnej trucizny(hamsters). Każdy ze skrzatów, wie jaki jest podstawowy poziom trucizny i ma to zapisane w swojej pamięci. Każdy skrzat wysyła wiadomość WANT_POISON = 7, informując pozostałych, że chce ubiegać się o truciznę. Skrzaty odbierają wiadomości i wysyłają OK_POISON = 8, gdy:

- nie ubiegają się o truciznę
- mają znacznik czasowy wyższy lub w przypadku równych znaczników mają wyższe id

Skrzat który otrzyma N-1 wiadomości OK_POISON po pobraniu trucizny wysyła kolejnym oczekującym procesom OK_POISON z liczbą trucizny, którą pobrał. Dzięki temu procesy wiedza ile trucizny się znajduje. Kolejne wchodzące procesy oprócz tego, że muszą mieć N-1 wiadomości potwierdzających sprawdzają również czy w zapasie będzie potrzebna ilość trucizny. Jeśli jej nie będzie będą oczekiwać na wiadomośc REQUEST_FINISHED = 9, która mówi nam o zakończeniu zlecenia i zwolnieniu pewnej liczby trucizny. Z kolei gdy burmistrz otrzyma Z(liczba zleceń), generuje kolejne zlecenia i wysyła ponownie KILL_REQUEST.

```
rec_ok_poison = 0; // licznik do zliczania ilości otrzymanych OK_POISON
<List> sent_ok_poison // tablica id, którym musimy wysłac OK_POISON po pobraniu
trucizny
temp_lamport = my_clock;

message.sender_id = my_id;
message.type=WANT_POISON;
message.lamport_clock = temp_lamport;

While(rec_ok_poison!=N-1){
        recv(message);
        temp_lamport = max(temp_lamport,message.lamport_clock)+1;
        if(message.type == OK_POISON) {
                rec_ok_poison++
                local_poison -= message.poison;
        }
        if(message.type == WANT_POISON){
                if(my_clock>message.lamport_clock or(my_clock ==
message.lamport_clock && my_id > message.sender_id) { send(message.sender_id,
OK_POISON) ;}
// tutaj bez wartości poison jezeli poison będzie puste to odejmiemy zwyczajnie 0,
poza tym wątki które wejdą przed nami nie obchodzi ile trucizny zabierzemy z puli
        else{ sent_ok_poison.append(sender.id);}
        if(message.type == REQUEST_FINISHED){poison+=message.poison}
        }
}
my_clock = lamport_clock;
//skrzat wychodzący z pętli
wait_until(my_request.hamsters>poison){
//czekaj na message =REQUEST_FINISHED
recv(message);
my_clock = max(my_clock,message.lamport_clock) + 1;
if(message.type == REQUEST_FINISHED){poison+=message.poison};
}
//pobiera truciznę
```

//wysyła OK_POISON

//Wykonuje zadanie w zależności od ilości chomików
//Wysyła wiadomości o zwolnieniu zasobów agrafki i trucizny a także o zakończeniu zlecenia i oczekuje ponownie na KILL_REQUEST.

Złożoność tej operacji wiadomości to także około liniowej N.

3. Skrzat po wykonanym zleceniu informuje Burmistrza o zakończeniu REQUEST_FINISHED=9 oraz numer zlecenia. Następnie czeka na sygnal KILL_REQUEST = 1.

4. Burmistrz gdy otrzyma informację o tym, że wszystkie zlecenia zostały zakończone tworzy nowe zlecenia i wysyła sygnał broadcastowy KILL_REQUEST = 1.

   //proces burmistrza

   recv(message)
   lamport_b = max(lamport_b, message.lamport_clock) + 1
   finished++

   if (finished == request_num){
   	create_new_request();
   	azerstaf();
   }

   send_broadcast(message);