Konstrukcja spójnego obrazu stanu globalnego – algorytmy

Plan wykładu

Celem wykładu jest przedstawienie niektórych algorytmów służących do konstrukcji obrazu spójnego stanu globalnego. Wykład obejmie przedstawienie następujących algorytmów: algorytm Chandy-Lamporta dla kanałów FIFO, algorytm Matterna stosujący zegary wektorowe, Lai-Yang dla kanałów non-FIFO, algorytm kolorujący procesy i wiadomości, oraz dwa algorytmy dla kanałów typu FC (Chandy-Lamporta i stosujący znaczniki BF i FF). Dla niektórych algorytmów zostanie omówiona ich złożoność czasowa oraz dowiedziona ich poprawność.

Założenia dotyczące środowiska przetwarzania:

Zostanie obecnie przedstawiona algorytm przeprowadzający konstrukcję spójnego obrazu stanu globalnego dla środowiska z kanałami FIFO.

Chandy i Lamport jako pierwsi przedstawili rozwiązanie problemu konstrukcji obrazu spójnego w środowisku rozproszonym. Ich algorytm wymaga przyjęcia kilku założeń dotyczących środowiska przetwarzania:

- niezawodne kanały zachowują uporządkowanie wiadomości (niezawodne kanały FIFO)
- stan reprezentowany jest w postaci złożenia lokalnych stanów procesów i stanów kanałów
- pełen asynchronizm komunikacji i przetwarzania
- brak zegara globalnego

Algorytm Chandy – Lamporta: Koncepcja (1)

- Pewien monitor Q_α (inicjator) spontanicznie podejmuje decyzje o zainicjowaniu *procesu konstrukcji* (*detekcji*) spójnego obrazu stanu globalnego.
- W pierwszej kolejności Q_α zapamiętuje stan lokalny skojarzonego z nim procesu aplikacyjnego P_α i wysyła wiadomość kontrolną (znacznik) typu MARKER, do wszystkich incydentnych monitorów.

Algorytm Chandy – Lamporta: Koncepcja (2)

- Każdy monitor Q_i po odebraniu znacznika z kanału $C_{j,i}$, sprawdza czy jest to pierwszy znacznik odebrany w danym procesie detekcji (tzn. czy stan procesu został już w tym procesie detekcji zapamiętany).
- Jeżeli jest to pierwszy znacznik, to Q_i zapamiętuje aktualny stan S_i procesu P_i, uznaje stan kanału $C_{j,i}$ za pusty, i propaguje znacznik przez wszystkie swoje kanały wyjściowe (wysłanie znacznika musi poprzedzić zdarzenie wysłania danym kanałem kolejnej, po zapamiętaniu stanu, wiadomości aplikacyjnej).

Algorytm Chandy – Lamporta: Koncepcja (3)

- Jeżeli stan procesu P_i został już wcześniej zapamiętany, to Q_i uznaje za stan kanału $C_{j,i}$ zbiór tych wszystkich wiadomości aplikacyjnych, które dotarły tym kanałem po zapamiętaniu stanu a przed otrzymaniem znacznika.
- Po odebraniu znaczników ze wszystkich kanałów wejściowych, monitor Q_i przesyła zapamiętany stan lokalny $procState_i$ procesu P_i oraz stan $chanState_i$ wszystkich kanałów wejściowych do monitora Q_β, konstruującego obraz stanu globalnego.
- W szczególności możliwe jest oczywiście, że $Q_\alpha = Q_\beta$.

Ilustracja działania algorytmu Chandy-Lamporta

Dla zilustrowania działania tego algorytmu przeanalizujemy przykład przedstawiony na slajdzie. Przyjęto na nim, że linia czerwona oznacza przesłanie wiadomości aplikacyjnej (na przykład M_1) natomiast liniami zielonymi oznaczono przesłanie znacznika między monitorami. Założono ponadto, że topologia przetwarzania reprezentuje graf w pełni połączony. Inicjatorem detekcji jest monitor $Q_\alpha = Q_1$. Na górnej części rysunku w wyniku zdarzenia σ_1^1 zachodzącego w Q_1 w zmiennej $procState_i$ zapamiętany zostaje stan lokalny S_1^1 procesu P_1, zmiennej $chanState_i$ nadawana jest wartość początkowa $\varnothing$, a znaczniki zostają wysłane do monitorów Q_2 oraz Q_3. Odebranie pierwszego znacznika przez Q_3 w wyniku zdarzenia σ_3^1 powoduje zapamiętanie stanu lokalnego S_3^1 i uznanie kanału wejściowego $C_{1,3}$ za pusty (podstawienie $recvMark_3[1]:=True$).

Z kolei wiadomość M_1, odebrana przez P_1 po zapamiętaniu stanu lokalnego, zostaje uwzględniona w stanie kanału $C_{3,1}$. Zajście zdarzenia σ_1^3 powoduje, że ostatecznie stan kanału $C_{3,1}$ będzie zawierał wiadomość M_1. Ostatecznie więc zostaje wyznaczony następujący stan spójny:

$$\Gamma^1 = \langle S_1^1, S_2^1, S_3^1, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\varnothing \rangle$$

Dla porównania stan wyznaczony w procesie detekcji na dolnej części rysunku wygląda następująco:

$$\Gamma^2 = \langle S_1^1, S_2^2, S_3^3, L_{1,2}=\varnothing, L_{1,3}=\varnothing, L_{2,1}=\varnothing, L_{2,3}=\varnothing, L_{3,1}=\{M_1\}, L_{3,2}=\{M_2\} \rangle.$$

Jak widać, mimo, że proces detekcji w obu przypadkach rozpoczęty został w tej samej chwili przez ten sam monitor, otrzymane konfiguracje spójne Γ^1 oraz Γ^2 są różne. Łatwo też zauważyć, że Γ^1 reprezentuje stan $\Sigma(\tau_b)$, podczas gdy Γ^2 reprezentuje stan, który nie wystąpił w rozpatrywanej realizacji (w żadnej chwili nie ma jednocześnie w kanałach $C_{3,1}$ i $C_{3,2}$ wiadomości). Ta różnica nie powinna zaskakiwać z uwagi na asynchronizm komunikacji i wynikający z tego niedeterminizm przetwarzania.

Algorytm Chandy – Lamporta (1)

Algorytm używa wiadomości trzech typów. Wiadomość typu PACKET jest zwykłą wiadomością aplikacyjną. Typ MARKER oznacza znacznik, podczas gdy STATE oznacza wiadomość zawierającą stan procesu (pole *procState*) oraz stan kanałów incydentnych z procesem (pole *chanState*).

Algorytm Chandy – Lamporta (2)

Wiadomość *msgIn* jest typu MESSAGE i oznacza wiadomość wysyłaną przez proces aplikacyjny. Zostaje ona opakowana i przesłana dalej w postaci *pcktOut* typu PACKET przez monitor procesu. Znacznik jest oznaczony jako *markerOut* zaś wiadomość przesyłająca stan jako *stateOut.*

Stan kanałów wejściowych procesu P_i, lista komunikatów odebranych od momentu zapamiętania stanu przez monitor Q_i do momentu otrzymania znacznika od Q_j zapisywany będzie w zmiennej $chanState_i$, zaś stan procesu w zmiennej $procState_i$. Tablica $recvMark_i$ dostarcza informacji, czy proces otrzymał znacznik od *j*-tego procesu (wartość elementu *n* równa *True*) i tym samym został wyznaczony stan kanału $C_{i,j}$. Zbiór kanałów wejściowych i wyjściowych procesu P_i opisywany jest odpowiednio przez $\mathcal{C}_i^{IN}$ oraz $\mathcal{C}_i^{OUT}$. Wreszcie wartość *True* zmiennej $involved_i$ oznacza, że proces zaangażował się już w konstrukcję obrazu stanu globalnego.

Algorytm Chandy – Lamporta (3)

W algorytmie używane będą dwie procedury. Pierwsza z nich, RECORDSTATE, zapisuje stan procesu, ustawia stan kanałów zapisany w tablicy $chanState_i[j]$ na $\varnothing$, zaznacza, że proces wziął już udział w wyznaczaniu obrazu i wysyła znacznik do wszystkich monitorów połączonych z procesem P_i za pomocą kanałów wejściowych.

Procedura SENDSTATE z kolei powoduje przesłanie stanu procesu P_i do monitora Q_β.

Algorytm Chandy – Lamporta (4)

Proces inicjator konstrukcji zapamiętuje własny stan i rozsyła znaczniki do incydentnych monitorów za pomocą procedury RECORDSTATE, ustawia pozycję α w tablicy $recvMark_i$ na $True$ oraz wysyła swój stan za pomocą procedury SENDSTATE.

Algorytm Chandy – Lamporta (5)

Kiedy zachodzi zdarzenie wysłania wiadomości przez proces P_i, wiadomość ta jest umieszczana w pakiecie i przesyłana między odpowiednimi monitorami.

Algorytm Chandy – Lamporta (6)

W przypadku otrzymania znacznika monitor zapisuje stan skojarzonego z nim procesu aplikacyjnego (o ile jeszcze tego wcześniej nie wykonał) i przesyła znacznik dalej (przy pomocy procedury RECORDSTATE), ustawia odpowiedni wpis w tablicy $recvMark$ na $True$. Jeżeli monitor otrzymał już znaczniki wszystkimi kanałami wejściowymi, rozsyła swój stan do incydentnych monitorów.

Algorytm Chandy – Lamporta (7)

Jeżeli monitor Q_i jest już zaangażowany w proces konstrukcji spójnego obrazu stanu globalnego, to otrzymując dowolną wiadomość aplikacyjną, przed przekazaniem jej do skojarzonego procesu aplikacyjnego dodaje ją do odpowiedniej pozycji zmiennej $chanState$.

Twierdzenie 8.1

Twierdzenie 8.1

Algorytm Chandy-Lamporta wyznacza w skończonym czasie konfigurację spójną.

Dowód

Lemat 8.1.1

Jeżeli co najmniej jeden proces zainicjuje algorytm, wszystkie procesy zapiszą swój stan lokalny w skończonym czasie.

Ponieważ każdy proces zapisuje stan i wysyła znacznik co najwyżej raz, algorytm się kończy w skończonym czasie. Jeżeli P_i jest procesem, który już zapisał stan, a P_j jest sąsiadem P_i, to P_j również zapisał stan. Wynika to z faktu, że znacznik wysłany przez P_i i P_j odebrany przez wymusza na nim zapisanie stanu, jeżeli wcześniej już tego nie dokonał. Ponieważ co najmniej jeden proces zainicjował algorytm, co najmniej jeden zapisał stan; a ponieważ zakładamy, że wszystkie procesy są połączone (pośrednio lub bezpośrednio) za pomocą sieci komunikacyjnej, implikuje to, że wszystkie procesy zapisały stan.

Należy obecnie wykazać, że wyznaczona konfiguracja jest spójna. Oznacza to, że nie może ona zawierać wiadomości w zapisanym stanie kanałów bądź zdarzeń odbioru wiadomości wysłanych w wyniku pewnego zdarzenia, które by nie zostało zapisane w obrazie stanu globalnego. Przyjmijmy, że Jeżeli proces zapisał swój stan, to wysłał również znacznik do wszystkich sąsiadów. Oznacza to, że każde zdarzenia wysłania przez P_i wiadomości M do innego procesu P_j po zapisaniu stanu musi być poprzedzone zdarzeniem wysłania znacznika - z własności FIFO kanałów komunikacyjnych oznacza to, że P_j otrzyma znacznik przed wiadomością M . Z algorytmu wynika, że otrzymania znacznika przez P_j wymusza na nim zapisanie stanu – a więc jest niemożliwe, by proces otrzymał tą wiadomość przed zapisaniem stanu. Oznacza to, że wyznaczona konfiguracja nie będzie zawierała żadnej wiadomości, której zdarzenie wysłania nie zostało zapisane w którymś z stanów lokalnych. a więc jest konfiguracją spójną.

Twierdzenie 8.2

Twierdzenie 8.2

Jeżeli proces konstrukcji konfiguracji spójnej rozpoczął się w chwili τ_b, a zakończył w chwili τ_e, to wyznaczona konfiguracja Γ, reprezentująca pewien stan globalny Σ, jest osiągalna ze stanu $\Sigma(\tau_b)$, a stan globalny $\Sigma(\tau_e)$ jest osiągalny ze stanu $\Sigma = \Gamma$. Tym samym:

$$\Sigma(\tau_b) \rightsquigarrow \Gamma \rightsquigarrow \Sigma(\tau_e)$$

Złożoność czasowa algorytmu Chandy-Lamporta

Przyjmujemy teraz, że graf zorientowany odpowiadający topologii przetwarzania rozproszonego scharakteryzowany jest przez następujące parametry :

m – liczba krawędzi grafu (jednokierunkowych kanałów komunikacyjnych),

d – średnica grafu,

l – długość najdłuższej ścieżki w grafie.

Zatem złożoność czasowa algorytmu Chandy-Lamporta wynosi *d* + 1, a złożoność komunikacyjna, w sensie liczby przesyłanych znaczników i pomijając zbieranie wiadomości przez Q_β, wynosi 2*m*.

Wyznaczanie obrazu spójnego: Kanały non-FIFO

W algorytmie Chandy-Lamporta przesyłane znaczniki pełniły podwójną rolę. Po pierwsze, dzięki zachowaniu uporządkowania FIFO pakietów, znaczniki określały pośrednio zbiór $outLog_{i,j}(\tau_i)$ wiadomości wysyłanych przez nadawcę P_i do P_j w chwili τ_i zapamiętywania stanu lokalnego procesu P_i. Zbiór $outLog_{i,j}(\tau_i)$ jest bowiem jednocześnie zbiorem $inLog_{i,j}(\tau_j')$ wiadomości odebranych przez P_j do chwili otrzymania znacznika (wiadomości wysłane przed znacznikiem dotrą przez nim do procesu docelowego), czyli $outLog_{i,j}(\tau_i) = inLog_{i,j}(\tau_j')$. Jeżeli więc stan procesu P_j zostanie zapamiętany w chwili $\tau_i \leq \tau_j$'. to stan kanału $C_{i,j}$ w chwili τ_j jest równy $outLog_{i,j}(\tau_i) \setminus inLog_{i,j}(\tau_j)$. Dodatkowo znaczniki przesyłane kanałami FIFO gwarantują oczywiście że $outLog_{i,j}(\tau_j) \subseteq inLog_{i,j}(\tau_i)$, a tym samym zapewniają spójność obrazu stanu globalnego.

Lai i Yang przedstawili algorytm, który nie wykorzystuje znaczników i nie wymaga by kanały były typu FIFO. W algorytmie zakłada się, że reprezentacje stanów lokalnych obejmują **historię komunikacji**, a więc odpowiednie zbiory wiadomości dotychczas wysłanych i odebranych. W efekcie, monitor Q_β konstruujący obraz spójny może łatwo wyznaczyć stany poszczególnych kanałów po otrzymaniu stanów lokalnych i historii komunikacji od wszystkich monitorów, jako różnicę $outLog_{i,j}(\tau_i) \setminus inLog_{i,j}(\tau_j)$; gdzie $\tau_j \leqslant \tau_j$.

Nie ma więc też potrzeby specjalnego odnotowywania wiadomości odebranych z danego kanału po zapamiętaniu stanu lokalnego (τ_j) a przed otrzymaniem znacznika (τ_j').

Algorytm Lai-Yang: Koncepcja (1)

Drugi pomysł zastosowany w tym algorytmie związany jest ze spostrzeżeniem, że jeśli po zapamiętaniu stanu proces P_i już nigdy nie wyśle wiadomości do P_j, to nie ma potrzeby przesłania do Q_j znacznika. Jeśli zaś wiadomości będą w dalszym ciągu wysyłane, to można do nich zawsze dołączyć jest do nich znacznik w formie etykiety, a tym samym odróżnić te pakiety od wszystkich pakietów wysłanych przed zapamiętaniem stanu.

Algorytm Lai-Yang: Koncepcja (2)

Wyróżnione pakiety ze znacznikiem przechwytywane są przez monitor Q_j odbiorcy i powodują najpierw zapamiętanie stanu procesu P_j, a dopiero w następnej kolejności przekazanie wiadomości aplikacyjnej procesowi P_j.

Żadna z wiadomości wysłanych przez P_i do P_j po zapamiętaniu stanu procesu P_i, nie zostanie odebrana przez P_j przed zapamiętaniem jego stanu. Oznacza to dalej, że spełniona będzie relacja:

$inLog_{i,j}(\tau_j) \subseteq outLog_{i,j}(\tau_i)$

która gwarantuje spójność konstruowanego obrazu stanu globalnego przetwarzania rozproszonego.

Przypisanie odpowiedniego koloru pakietowi oznacza:

- *White* – pakiet bez znacznika
- *Red* – dołączenie znacznika do wiadomości

Analogicznie, kolory przypisane są procesom (monitorom):

- *White* – proces nie zapamiętał jeszcze stanu
- *Red* – stan procesu został już zapamiętany

Ogólna idea algorytmu jest następująca:

- Każdy proces jest postrzegany przez monitor początkowo jako *White*, a monitor zmienia jego kolor na *Red* po zapamiętaniu stanu procesu.
- Każdej wiadomości wysłanej przez proces koloru *White* przypisany jest kolor *White*, a każdej wysłanej przez proces *Red* – kolor *Red*.

Stan procesu koloru *White* można zapamiętać w dowolnej chwili, lecz koniecznie przed odebraniem wiadomości koloru *Red*.

Algorytm Lai-Yang (1)

Algorytm używa dwóch typów wiadomości. Wiadomości aplikacyjne przesyłane są w pakietach typu PACKET, które dodatkowo zawierają etykietę wiadomości w polu *colour*, będącą jedną z wartości *White* albo *Red*. Drugim typem jest STATE zawierający pole *procState*, określające stan lokalny procesu, *sentLog* z zbiorem wiadomości aplikacyjnych wysłanych do chwili zapamiętania stanu procesu, oraz *recvLog* z zbiorem wiadomości aplikacyjnych odebranych do chwili zapamiętania stanu procesu.

Algorytm Lai-Yang (2)

Wiadomość *msgIn* to wiadomość aplikacyjna wypakowywana z odebranego przez monitor pakietu, która zostanie dostarczona procesowi aplikacyjnemu. Wysyłane wiadomości są opakowywane i wysyłane jako *pcktOut*. Stan procesu przesyłany jest w wiadomości *stateOut*. Zakładamy, że stan procesu P_i określony jest w zmiennej $procState_i$. Zmienna $procColour_i$ zawiera kolor monitora, początkowo ustawiony na *White*. Zmienna $sentLog_i$ określa zbiór wiadomości aplikacyjnych wysłanych przez P_i do chwili zapamiętania stanu procesu. Zmienna $outLog_i$ określa zbiór wiadomości wysłanych przez P_i do chwili obecnej. Zbiór wiadomości aplikacyjnych odebranych przez P_i do chwili zapamiętania stanu procesu przechowywany jest w zmiennej $recvLog_i$, a zbiór wiadomości aplikacyjnych odebranych przez P_i do chwili obecnej w $inLog_i$.

Pakiet typu $dummyOut$ to wiadomość oznaczona specjalnym znacznikiem inicjującym, przenoszący pustą wiadomość aplikacyjną.

Algorytm Lai-Yang (3)

Procedura RECORDSTATE zapamiętuje stan procesu, przypisując stan lokalny S_i do zmiennej $procState_i$, a zawartość zmiennych $outLog_i$ i $inLog_i$ przypisując do $sentLog_i$ i $recvLog_i$. Procedura SENDSTATE służy do przesłania zapisanego stanu procesu P_i do monitora Q_β.

Algorytm Lai-Yang (4)

Konstrukcja obrazu spójnego stanu globalnego rozpoczynana jest przez monitor Q_α. Zapisuje on stan skojarzonego z sobą procesu aplikacyjnego, zmienia kolor procesu na Red i wysyła pusty znacznik $dummyOut$ o kolorze Red do wszystkich procesów. Następnie rozsyła również swój stan.

Algorytm Lai-Yang (5)

Wysyłając jakąkolwiek wiadomość aplikacyjną, monitor zapamiętuje ją w zmiennej $outLog_i$, oraz oznacza ją kolorem procesu.

Algorytm Lai-Yang (6)

Otrzymując pakiet o kolorze Red, monitor Q_i procesu aplikacyjnego P_i o kolorze $White$ zmienia jego kolor na Red i rozsyła jego stan. Przed dostarczeniem wiadomości do procesu P_i monitor zapamiętuje ją w zmiennej $inLog_i$.

Algorytm Lai-Yang: Złożoność czasowa

Jeżeli przez *d* oznaczymy średnicę grafu zorientowanego odpowiadającego topologii przetwarzania rozproszonego, a przez *n* liczbę wierzchołków tego grafu, to złożoność czasowa algorytmu Lai-Yanga wynosi *d*, a złożoność komunikacyjna, w sensie liczby przesyłanych znaczników, wynosi *n*-1 (dla grafu pełnego).

Zaniechanie wysyłania znaczników, a jedynie dopisywanie pewnej informacji do wiadomości aplikacyjnych (kolorowanie pakietów), grozi tym, że:

- w wypadku braku komunikacji na poziomie aplikacyjnym stany pewnych procesów nigdy nie zostaną zapamiętane
- konfiguracja globalna nigdy nie zostanie wyznaczona

Aby takiej sytuacji zapobiec, inicjator może wysłać specjalny pakiet sterujący z pustym polem danych, do monitorów wszystkich procesów tworzących przetwarzanie. Przy takim rozszerzeniu algorytm Lai-Yanga wyznacza spójny obraz w skończonym czasie.

Twierdzenie 8.3

Twierdzenie 8.3

Konfiguracja wyznaczona przez algorytm Lai-Yang jest konfiguracją spójną.

Dowód

Przyjmijmy, że istnieje wiadomość M wysłana z P_i do P_j po fakcie zapisania stanu procesu P_i. To oznacza, że znacznik związany z wiadomością M musi mieć wartość Red, co wymusza na P_j zapisanie stanu lokalnego najpóźniej w momencie odebrania M. Tak więc, stan zapisany przez P_j nie zawiera zdarzenia odbioru M.

Niech pakiet sterujący będzie wysyłany kanałami odpowiadającymi krawędziom drzewa rozpinającego o wysokości *d*, grafu reprezentującego topologię przetwarzania. Pomijając fazę zbierania informacji z

poszczególnych węzłów i koszt zmiany koloru pakietów, to złożoność czasowa i komunikacyjna algorytmu Lai-Yanga wynosi odpowiednio *d* i *n*-1.

Można dowieść, że jeżeli kanały komunikacyjne nie są FIFO oraz nie używa się znaczników dołączanych do wiadomości aplikacyjnych, konstrukcja obrazu stanu globalnego wymaga zatrzymania przetwarzania. W przypadku kanałów FIFO taki algorytm oczywiście istnieje (przykładem jest na przykład algorytm Chandy-Lamporta)

Algorytm stosujący zegary wektorowe

Rozważamy system, w którym kanały są kanałami *nonFIFO*.

Algorytm wyznaczający stan globalny dla takiego systemu wywodzi się z koncepcji wyznaczania stanu globalnego z użyciem zegara czasu rzeczywistego.

Ogólna koncepcja:

- określenie momentu czasu τ_s odpowiadającego przyszłości, w którym zostaną zapamiętane stany lokalne wszystkich procesów.
- po wyznaczeniu stanów lokalnych w momencie τ_s, monitory przesyłają informację o stanach lokalnych do monitora konstruującego obraz globalny.

W celu adaptacji powyższej koncepcji do asynchronicznego systemu rozproszonego, proces inicjatora określa wektorowy czas wirtualny momentu zapamiętania w przyszłości stanu lokalnego procesu.

Twierdzenie 8.4

Wykorzystywana jest tu następująca właściwość zegarów wektorowych:

Twierdzenie 8.4

W chwili, gdy uaktualniony został zegar wektorowy procesu P_i, zachodzi:

$$\nexists P_j :: P_j \in \mathcal{P} :: vClock_i < vClock_j$$

Powyższa właściwość oznacza w praktyce, że zmiana wektorowego zegara lokalnego w pewnym procesie P_i określa czas wirtualny, który z perspektywy wszystkich pozostałych procesów P_j ($i \neq j$) z pewnością nie jest czasem odnoszącym się do przeszłości.

Algorytm Matterna (1)

Algorytm wykorzystuje pięć różnych typów komunikatów. Typ PACKET służy do przenoszenia wiadomości aplikacyjnych i posiada pole *vClock*, które jest wektorową etykietą czasową pakietu. Typ komunikatu CONTROL zawiera pole *vRecordClock* będące polem zadanego czasu zapamiętania stanu.

Algorytm Matterna (2)

Komunikaty typu PROC_STATE zawierają stan procesu w polu *procState*. Stan kanału przesyłany jest w polu *chanState* wiadomości typu CHAN_STATE. Wreszcie, wiadomości typu ACK są prostymi potwierdzeniami nie zawierającymi żadnych pól.

Algorytm Matterna (3)

Wiadomość *msgIn* to wiadomość aplikacyjna wypakowywana z odebranego przez monitor pakietu, która zostanie dostarczona procesowi aplikacyjnemu. Wysyłane wiadomości są opakowywane i wysyłane jako *pcktOut*. Istnieje specjalny pusty pakiet *dummyOut* zawierająca pusty komunikat aplikacyjny, wysyłana podczas rozpoczęcia algorytmu. Pakiety *ctrlOut, ackIn*, *ackOut* to odpowiednio pakiety kontrolne i potwierdzenia. Pakiety *procStateOut* oraz *chanStateOut* służą do przesłania stanu kanałów i procesu. Zmienna $vClock_i$ jest wektorowym zegarem logicznym procesu P_i. Zmienna $vRecordClock_i$ reprezentuje logiczny czas zadanego momentu zapamiętania obrazu spójnego.

Wreszcie, stan lokalny procesu P_i jest zapamiętany w zmiennej $procState_i$. Zmienna $recorded_i$ służy do określania, czy został już zapamiętany stan procesu P_i.

Algorytm Matterna (4)

Dla uproszczenia, będziemy tu zakładać, że $Q_\alpha = Q_\beta$.

- Monitor inicjatora zwiększa pozornie wartość swojego zegara wektorowego i tym samym wyznacza moment z jego perspektywy w przyszłości
- Informuje o tym wszystkie pozostałe monitory, wstrzymując jednocześnie postęp lokalnego czasu wirtualnego (blokuje postęp przetwarzania aplikacyjnego)

Dopiero po upewnieniu się, że wszystkie monitory znają wyznaczony moment w przyszłości, uwalniane jest przetwarzanie aplikacyjne, a więc również zegar lokalny. Monitor Q_α zapisuje wtedy stan skojarzonego procesu aplikacyjnego i rosyła pustą wiadomość *dummyOut*, zaznaczając jednocześnie, że stan procesu P_α został już zapisany.

Algorytm Matterna (5)

Otrzymując wiadomość kontrolną monitor Q_i zapamiętuje w zmiennej moment $vRecordClock_i$ wyznaczenia stanu lokalnego, a następnie odsyła potwierdzenie. Wysyłając wiadomość modyfikuje zegar wektorowy w zwykły sposób.

Algorytm Matterna (6)

Otrzymując pakiet aplikacyjny monitor Q_i modyfikuje zegar wektorowy w zwykły sposób. Jeżeli monitor zapisywał już stan aplikacyjny, a etykieta odebranego pakietu jest wcześniejsza od momentu wyznaczania stanu zapisanego w $vRecordClock_i$, monitor wysyła informację o otrzymaniu tego pakietu do inicjatora konstrukcji spójnego obrazu stanu globalnego.

Algorytm Matterna (7)

W wyniku dalszej komunikacji między procesami aplikacyjnymi, bądź w wyniku przesłania dodatkowych wiadomości synchronizujących między monitorami, każdy z procesów osiągnie w końcu, lub będzie usiłował przekroczyć, wyznaczony moment $vRecordClock_i$. Przed przekroczeniem wartości τ_s, zapamiętany zostanie stan lokalny procesu.

Kolekcja takich stanów tworzy konfigurację spójną.

Algorytm kolorujący procesy i wiadomości: Koncepcja (1)

Wadą przedstawionego algorytmu Matterna jest zawieszanie przetwarzania aplikacyjnego procesu P_α do momentu uzyskanie przez monitor Q_α potwierdzeń odebrania wiadomości kontrolnej zawierającej żądany moment zapamiętania w przyszłości stanu lokalnego od wszystkich pozostałych monitorów.

Problemu tego można uniknąć wprowadzając dodatkową $n + 1$ pozycję w wektorze czasu, której wartość jest zwiększana tylko przez wyróżniony monitor $Q_{n+1} = Q_\alpha$ nie skojarzony z żadnym procesem aplikacyjnym. Należy zauważyć, że w nowym wektorze czasu $vClock_i$' istotne znaczenie będzie miała teraz tylko pozycja n +1, a wszystkie pozostałe mogą być pominięte. W konsekwencji można też pominąć całą fazę rozgłaszania wartości $vRecordClock$. W tym wypadku procesy będą zapamiętywać stan lokalny, gdy uzyskają wektor $vClock_i$' z nową, powiększoną wartością pozycji $n + 1$.

Algorytm można uprościć dalej zauważając, że w kontekście detekcji stanu globalnego znaczenie ma w istocie tylko zmiana stanu na pozycji n+1, i stąd dwa stany tej pozycji są wystarczające dla rozróżnienia kolejnych faz wyznaczania obrazu globalnego:

- *White* – przed zapamiętaniem stanu lokalnego procesu
- *Red* – po zapamiętaniu stanu lokalnego.

Podobnie, kolory można przypisać pakietom. Kolor *White* i *Red* pakietu oznacza, że został on wysłany, odpowiednio, przed lub po zapamiętaniu stanu lokalnego procesu. Monitory procesów *White* wysyłają tylko pakiety *White*, a monitory procesów *Red* – tylko pakiety koloru *Red*.

Każdy proces pierwotnie ma kolor *White*, a staje się *Red* po zapamiętaniu jego stanu lokalnego a przed przekazaniem mu pierwszej wiadomości z pakietem koloru *Red*.

Inicjator detekcji stanu globalnego zapamiętuje stan lokalny procesu, staje się *Red* i wysyła pusty pakiet *dummyOut* koloru *Red* do wszystkich monitorów.

Poprawność algorytmu wynika z faktu, że nie istnieje pakiet wysłany przez monitor *Red*, którego wiadomość przekazana zostałaby procesowi aplikacyjnemu koloru *White*, gdyż taki pakiet spowodowałby zmianę koloru procesu odbierającego na *Red* przed przekazaniem wiadomości.

Zakłada się, że inicjator nie inicjuje współbieżnie wielu detekcji, dlatego wprowadzenie większej liczby kolorów (wartości elementu czasu wektorowego) nie jest konieczne.

Algorytm kolorujący procesy i wiadomości: Koncepcja (2)

W tym momencie łatwo zauważyć, że algorytm staje się praktycznie identyczny jak wcześniej przedstawiony algorytm Lai-Yang. Różnicą jest jedynie prostsza reprezentacja stanu lokalnego – gdzie nie ma potrzeby pamiętania historii komunikacji, co jednak pociąga za sobą konieczność wprowadzenia dodatkowego mechanizmu wyznaczania stanów kanałów komunikacyjnych.

Wiadomości będące w wyznaczonym obrazie stanu globalnego w kanałach, to wiadomości w pakietach koloru *White* odebrane przez monitor koloru *Red*.

Za każdym razem, gdy monitor otrzymuje tego typu pakiet, przesyła zawartą w nim wiadomość do inicjatora

Jedynym problemem w tym przypadku jest wyznaczenie momentu zakończenia konstrukcji obrazu stanu globalnego – tzn. podjęcia przez inicjatora decyzji, że zebrany dotychczas stan jest stanem kompletnym i żaden z monitorów nie prześle w przyszłości pakietu koloru *White*.

Problem ten, znany jako *problem detekcji zakończenia przetwarzania rozproszonego.*

Algorytm kolorujący procesy i wiadomości (1)

Tak zmodyfikowany algorytm używa trzech zamiast pięciu typów komunikatów. Struktura tych typów jest identyczna jak w poprzednio, z jednym wyjątkiem: pakiet typu PACKET zamiast etykiety czasu wektorowego posiada pole *colour* oznaczającego kolor pakietu (*Red, White*).

Algorytm kolorujący procesy i wiadomości (2)

Również liczba używanych zmiennych jest mniejsza niż w poprzednim algorytmie, a główna różnica wynika z braku $vRecordClock_i$ oraz $vClock_i$ oraz wprowadzenie zmiennej zawierającej kolor procesu, $procColour_i$.

Algorytm kolorujący procesy i wiadomości (3)

Inicjator rozpoczyna konstrukcję obrazu spójnego zapamiętując stan S_α, zmieniając kolor procesu na *Red* oraz wysyłając pusty pakiet aplikacyjny o kolorze *Red* do pozostałych procesów.

Algorytm kolorujący procesy i wiadomości (4)

Każda wysyłana wiadomość zawiera znacznik określający kolor nadawcy tej wiadomości.

Algorytm kolorujący procesy i wiadomości (5)

Jeżeli proces skojarzony z monitorem Q_i posiada kolor *Red*, to otrzymanie pakietu aplikacyjnego w kolorze *White* powoduje przesłanie informacji o tym do monitora Q_α.

Algorytm kolorujący procesy i wiadomości (6)

Z kolei otrzymania pakietu o kolorze Red gdy proces skojarzony z monitorem Q_i posiada kolor $White$ powoduje zmianę koloru procesu, oraz zapamiętanie i przesłanie informacji o jego stanie do monitora Q_α.

Kanały typu FC

Przypomnijmy, że kanały *FC* umożliwiają dobór operacji komunikacji i tym samym własności przysyłanych wiadomości, stosownie do wymaganego zastosowania.

- Użycie wyłącznie operacji $send^t$ powoduje, że kanał typu *FC* ma własności kanału *FIFO*.
- Użycie tylko operacji $send^o$, to kanał typu *FC* równoważny byłby kanałowi *nonFIFO*.

Tak więc kanały typu *FC* są mechanizmem elastycznym oferującym dobór operacji komunikacji i tym samym różne stopnie współbieżności komunikacji.

Możliwa jest modyfikacja algorytmu Chandy-Lamporta, polegająca na zastąpieniu znacznika przysłanego kanałem *FIFO*, przez znacznik typu *TF* – TMARKER. Poza tym algorytm nie różni się istotnie od pierwowzoru.

Algorytm Chandy-Lamporta dla kanałów FC (1)

Typy komunikatów używane w przedstawionym algorytmie nie różnią się od typów używanych w wersji algorytmu dla kanałów FIFO. Różnica polega na zastąpieniu typu MARKER typem TMARKER będącego komunikatem typu *TF*.

Algorytm Chandy-Lamporta dla kanałów FC (2)

Zmienne używane w algorytmie posiadają znaczenie analogiczne do wcześniej przedstawianego algorytmu Chandy-Lamporta dla kanałów FIFO.

Algorytm Chandy-Lamporta dla kanałów FC (3)

Procedura RECORDSTATE nie różni się (poza używaniem znacznika TMARKER zamiast MARKER) od wersji dla kanałów FIFO.

Algorytm Chandy-Lamporta dla kanałów FC (4)

Inicjator Q_α zapamiętuje spontanicznie stan skojarzonego z nim procesu aplikacyjnego P_α i wysyła wiadomość kontrolną (znacznik) typu TMARKER do wszystkich incydentnych monitorów. W odróżnieniu od wersji algorytmu dla kanałów FIFO, nie jest wysyłany stan za pomocą procedury SENDSTATE.

Algorytm Chandy-Lamporta dla kanałów FC (5)

Monitory Q_i po odebraniu znacznika z kanału $C_{j,i}$ sprawdzają, czy stan lokalny został już wcześniej zapamiętany.

Jeśli okazuje się, że jest to pierwszy znacznik TMARKER odebrany w danym procesie konstrukcji obrazu spójnego, to stan lokalny jest zapamiętany przed dopuszczeniem do wysłania kolejnej wiadomości aplikacyjnej, znacznik typu TMARKER wysyłany jest poprzez wszystkie kanały wyjściowe. Stan kanału $C_{j,i}$ przyjmuje się przy tym jako zbiór pusty. Jeżeli odebrano już znaczniki TMARKER z wszystkich kanałów wejściowych, rozsyłany stan jest stan procesu i kanałów do wszystkich pozostałych monitorów.

Algorytm Chandy-Lamporta dla kanałów FC (6)

Podczas odbioru wiadomości aplikacyjnej następuje sprawdzenie, czy stan lokalny był już zapamiętany. Jeżeli tak, to wszystkie wiadomości odebrane między momentem zapamiętania stanu a momentem otrzymania znacznika są włączane do zbioru określającego stan kanału wejściowego $C_{j,i}$.

Algorytm z znacznikami BF i FF: Koncepcja

Rozwinięcie poprzedniego algorytmu opiera się na spostrzeżeniu, że do utworzenia spójnego obrazu przetwarzania rozproszonego wystarczy by stany lokalne zapamiętane były w wyniku odebrania znaczników typu *BF* (BMARKER).

Znacznik ten nie może być wyprzedzony przez żadną wiadomość, i stąd pakiety wysłane po zapamiętaniu stanu i po wysłaniu znacznika typu *BF* dotrą do monitora docelowego z pewnością po zapamiętaniu stanu lokalnego adresata. Nie zaistnieje więc sytuacja prowadząca do niespójnej konfiguracji, w której wiadomość nie uwzględniona w stanie lokalnym przez nadawcę (wysłana po zapamiętaniu stanu) zostanie uwzględniona w stanie lokalnym odbiorcy (odebrana przed zapamiętaniem stanu).

Znacznik typu *FF*, FMARKER, wysłany kanałem zaraz po zapamiętaniu stanu lokalnego procesu, nie wyprzedzi pakietów wysłanych przed nim. Jeżeli zatem monitor będzie zapamiętywał wszystkie wiadomości otrzymane danym kanałem od momentu zapamiętania stanu do momentu odebrania znacznika FMARKER, to zbiór tych wiadomości obejmie wszystkie wiadomości tworzące stan kanału (wysłane przed znacznikiem a odebrane po zapamiętaniu stanu procesu docelowego) oraz wiadomości wysłane po znaczniku FMARKER, które znacznik ten wyprzedziły.

Aby wyróżnić interesujące nas wiadomości tworzące stan kanału $C_{j,i}$ w obrazie spójnym, wystarczy dołączyć do wszystkich wiadomości aplikacyjnych oraz znaczników FMARKER, etykietę określającą numer sekwencyjny ostatnio wysłanego komunikatu.

Algorytm z znacznikami BF i FF (1)

W algorytmie tym wiadomości aplikacyjne typu PACKET zawierają pole *seqNo* określające numer sekwencyjny ostatnio komunikatu. Znaczniki FMARKER, będące komunikatami typu FF, zawierają pole *seqNoPred* określające numer sekwencyjny poprzedniego pakietu. Znaczniki typu BMARKER to komunikaty typu BF. Wreszcie pakiet typu STATE posiada identyczne znaczenie jak poprzednio.

Algorytm z znacznikami BF i FF (2)

Algorytm używa nieco innego zestawu zmiennych niż w wersji przeznaczonej dla kanałów FIFO. Znaczniki *fMarkerOut* oraz *bMarkOut* oznaczają odpowiednio pakiety typu FMARKER oraz BMARKER.

Dodatkowo dodane są dwie zmienne: $seqNo_i$, która zawiera numer sekwencyjny ostatnio wysłanego przez monitor Q_i pakietu oraz $pcktBuf_i$, która zapamiętuje zbiór pakietów odebranych od momentu zapamiętania stanu przez monitor Q_i do momentu otrzymania znacznika FMARKER od Q_j.

Algorytm z znacznikami BF i FF (3)

Procedura RECORDSTATE zapamiętuje stan lokalny procesu S_i, inicjuje wszystkie elementy tablicy $pcktBuf_i$ na $\varnothing$, zaznacza, że proces P_i wziął już udział w przetwarzaniu i następnie wysyła dwa znaczniki: *bMarkOut* i zaraz potem *fMarkOut*.

Algorytm z znacznikami BF i FF (4)

Inicjator konstrukcji obrazu spójnego stanu globalnego Q_α spontanicznie zapisuje stan za pomocą procedury RECORDSTATE.

Monitor Q_i , który jeszcze nie zapisał swojego stanu, wywołuje funkcję RECORDSTATE (zapisując stan procesu i rozsyłając do sąsiadów znaczniki typu BMARKER i FMARKER) po otrzymaniu znacznika typu BMARKER.

Algorytm z znacznikami BF i FF (5)

Monitor Q_i, który już zapisał swój stan (zmienna $involved_i$ przyjmuje wartość $True$) otrzymując zwykły pakiet aplikacyjny od monitora Q_j od którego nie otrzymano dotąd znacznika FMARKER, dopisuje otrzymany pakiet do zbioru na odpowiedniej pozycji tablicy $pcktBuf_i$.

Algorytm z znacznikami BF i FF (6)

Otrzymanie znacznika typu FMARKER nadesłanego przez monitor Q_j powoduje zapisanie tego faktu w tablicy $recvMark_i$ poprzez nadanie odpowiedniemu elementowi wartości $True$, a odpowiedniej *j*-tej pozycji tablicy $chanState_i$ przypisywana jest wartość zbioru pustego. Następnie dodawane są do niej wszystkie takie pakiety z tablicy *pcktBuf*, których numer sekwencyjny jest nie większy, niż numer sekwencyjny zawarty w nadesłanym znaczniku.

Algorytm z znacznikami BF i FF (7)

Jeżeli znacznik typu FMARKER otrzymano już wszystkimi kanałami wejściowymi, rozsyłany jest stan procesu.

Algorytm z znacznikami BF i FF (8)

Wysyłając wiadomość aplikacyjną, monitor Q_i inkrementuje zawartość licznika $seqNo_i$ oraz umieszcza go w polu *data* pakietu *pcktOut*.