Plan wykładu

Celem wykładu jest zaznajomienie studenta z problematyką detekcji zakleszczenia. Wykład obejmie omówienie podstawowych pojęć związanych z przedstawianą tematyką. Przedstawione zostaną definicje zakleszczenia dla różnych modeli żądań. Następnie omówiona zostanie klasyfikacja problemów detekcji zakleszczenia. Na zakończenie przedstawione zostaną trzy przykładowe algorytmy detekcji zakleszczenia dla modeli OR, AND i „k spośród r".

Wprowadzenie

Jak wiadomo, procesy tworzące przetwarzanie rozproszone komunikują się ze sobą za pomocą mechanizmu wymiany wiadomości, realizując wspólny cel przetwarzania. Przykładowo, procesy zarządzające elementami rozległej sieci komputerowej kooperują w celu efektywnego, niezawodnego i bezpiecznego rozdziału takich zasobów jak: bufory łączy komunikacyjnych, szerokość udostępnionego pasma transmisji, tablice wyboru trasy, tablice konwersji nazw itp. Tym samym, jedne procesy wysyłają komunikaty zawierające żądania przydziału pewnych zasobów, inne - w odpowiedzi - przesyłają ewentualnie komunikaty potwierdzające przydział żądanych zasobów.

Nieformalna definicja problemu

Jeżeli żądany zasób jest niezbędny dla kontynuacji działania, proces żądający musi czekać w stanie pasywnym aż do momentu nadejścia komunikatu potwierdzającego przydział lub zawierającego żądany zasób (obiekt). W przetwarzaniu rozproszonym może zatem w ogólności wystąpić sytuacja, w której wszystkie procesy pewnego niepustego zbioru procesów oczekują na wiadomości (potwierdzające na przykład przydział zasobów) od innych procesów tego właśnie zbioru. Stan taki nazywany jest **zakleszczeniem rozproszonym** (ang. *distributed deadlock*). Wystąpienie zakleszczenia wstrzymuje trwale procesy zakleszczone i tym samym blokuje będące w ich posiadaniu zasoby. Jeśli więc stan taki nie zostanie wykryty i nie będą podjęte odpowiednie działania, to procesy zakleszczone i blokowane przez nie zasoby mogą spowodować wstrzymanie innych procesów, czy wreszcie doprowadzić do blokady całego systemu rozproszonego. Stąd też wynika znaczenie problemu detekcji zakleszczenia rozproszonego. Wobec asynchronizmu komunikacji i przetwarzania oraz braku wspólnego zegara, rozwiązanie tego problemu jest trudne i wymaga rozpatrzenia wielu subtelnych kwestii.

Procesy aktywne i pasywne

W celu precyzyjnego sformułowania problemu detekcji zakleszczenia przetwarzania rozproszonego przyjmiemy, że w każdej chwili proces jest w jednym z dwóch stanów: aktywnym albo pasywnym. Przypomnijmy, że aktywny proces może realizować przetwarzanie wykonując operacje odpowiadające zajściu zdarzeń wewnętrznych i komunikacyjnych.

Warunek uaktywnienia

Ponadto, w dowolnej chwili aktywny proces P_i może zmienić swój stan na pasywny, definiując jednocześnie zbiór warunkujący $\mathcal{D}_i$ (będący sumą zbiorów oczekiwanych nadawców wszystkich dopuszczalnych zdarzeń odbioru) oraz warunek uaktywnienia wyrażony przez predykat: $ready_i(\mathcal{X}) \equiv (\mathcal{P}_i^A \supseteq \mathcal{X}) \wedge activate_i(\mathcal{X})$. Po spełnieniu warunku uaktywnienia, proces P_i zmienia swój stan na aktywny i w sposób atomowy (niepodzielny) pobiera z kanałów wejściowych wszystkie te wiadomości, których nadejście doprowadziło do spełnienia warunku uaktywnienia. Wprowadzimy jeszcze następujące oznaczenia:

$passive_i$ – zmienna logiczna (predykat) przyjmująca wartość $True$ wtedy i tylko wtedy, gdy proces P_i jest pasywny;

$available_i$ – tablica [1…n] zmiennych logicznych procesu P_i skojarzona z wiadomościami dostępnymi;

$available_i[j]$ – j-ty element tablicy $available_i$ przyjmujący wartość $True$ wtedy i tylko wtedy, gdy dla P_i jest dostępna wiadomość wysłana przez P_j;

$in\text{-}transit_i$ – tablica zmiennych logicznych procesu P_i skojarzona z wiadomościami transmitowanymi;

$in\text{-}transit_i[j]$ – j-ty element tablicy $in\text{-}transit_i$ przyjmujący wartość $True$ wtedy i tylko wtedy, gdy wiadomość wysłana przez P_j do P_i należy do $L_{j,i}{}^T$, a więc jest transmitowana i nie jest jeszcze dostępna.

Niech ponadto:

$$\mathcal{AV}_i = \{P_j : available_i[j] = True\}, \quad \mathcal{IT}_i = \{P_j : in\text{-}transit_i[j] = True\}.$$

Korzystając z powyższych oznaczeń, przedstawimy teraz formalne definicje zakleszczeń odpowiadające różnym modelom żądań: modelowi jednostkowemu, AND, OR, k spośród n, OR-AND, oraz modelowi predykatowemu. Dla uproszczenia, będziemy mówili o zakleszczeniu w określonym modelu, lub o modelu zakleszczenia.

Definicja problemu

Przez $deadlock(\mathcal{B})$ oznaczamy predykat stwierdzający, że w danej chwili τ, niepusty zbiór procesów $\mathcal{B}$ jest zbiorem procesów zakleszczonych.

Zakleszczenie w modelu *jednostkowym*

W modelu jednostkowym warunkiem uaktywnienia pasywnego procesu P_i jest przybycie wiadomości od ściśle określonego, jednego nadawcy. Tak więc dla każdego zbioru warunkującego $\mathcal{D}_i$, $|\mathcal{D}_i| = 1$. Wówczas:

$$\begin{aligned} &deadlock(\mathcal{B}) \equiv \\ &(\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\ &(\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge |\mathcal{D}_i| = 1 \wedge \\ &\qquad (\exists P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg in\text{-}transit_i[j] \wedge \neg available_i[j])))) \end{aligned}$$

Zakleszczenie w modelu *AND*

W modelu AND proces pasywny P_i staje się aktywny, jeżeli dotarły do niego wiadomości od każdego z procesów tworzących jego zbiór warunkujący $\mathcal{D}_i$. Wówczas:

$$\begin{aligned} &deadlock(\mathcal{B}) \equiv \\ &(\mathcal{B} \subseteq \mathcal{P}) \wedge (\mathcal{B} \neq \emptyset) \wedge \\ &(\forall P_i :: P_i \in \mathcal{B} :: (passive_i \wedge \\ &\qquad (\exists P_j :: P_j \in \mathcal{D}_i \cap \mathcal{B} :: (\neg in\text{-}transit_i[j] \wedge \neg available_i[j])))) \end{aligned}$$

Zakleszczenie w modelu *OR*

W modelu OR do uaktywnienia procesu P_i wystarczy jedna wiadomość od któregokolwiek z procesów jego zbioru warunkującego $\mathcal{D}_i$. Dlatego:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&(\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&(\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \ \wedge \\
&\qquad\qquad \mathcal{D}_i \subseteq \mathcal{B} \ \wedge \\
&\qquad\qquad (\forall\ P_j :: P_j \in \mathcal{D}_i :: (\neg\ in\text{-}transiti[j] \wedge \neg\ availablei[j]\ ))))
\end{aligned}
$$

Zakleszczenie w podstawowym modelu *k spośród r*

W podstawowym modelu k spośród r, z pasywnym procesem P_i skojarzony jest zbiór warunkujący $\mathcal{D}_i$, liczba naturalna k_i, $1 \leqslant k_i \leqslant |\mathcal{D}_i|$, oraz liczba naturalna $r_i = |\mathcal{D}_i|$. W modelu tym proces P_i staje się aktywny wówczas, gdy uzyska wiadomości od co najmniej k_i różnych procesów ze zbioru warunkującego $\mathcal{D}_i$.

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&(\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&(\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \wedge \\
&\qquad\qquad (\exists\ \mathcal{B}_i :: \mathcal{B}_i \subseteq \mathcal{D}_i \cap \mathcal{B} :: \\
&\qquad\qquad\quad (\ |\ \mathcal{D}_i \setminus \mathcal{B}_i\ | < k_i \wedge \\
&\qquad\qquad\qquad (\forall P_j :: P_j \in \mathcal{B}_i :: (\neg\ in\text{-}transit_i[j] \wedge \neg\ available_i[j]\ ))))))
\end{aligned}
$$

Definicja powyższa oznacza, że dla każdego procesu P_i można znaleźć zbiór procesów $\mathcal{B}_i$, od których nie jest możliwe otrzymanie wiadomości ($\mathcal{B}_i \subseteq \mathcal{B}$ i jednocześnie $(\forall P_j :: P_j \in \mathcal{B}_i :: (in\text{-}transit_i[j] \wedge \neg\ available_i[j]))$. Tak więc, P_i potencjalnie otrzyma co najwyżej $|\ \mathcal{D}_i \setminus \mathcal{B}_i\ |$ wiadomości, co jednak nie wystarcza do uaktywnienia, gdyż $|\mathcal{D}_i \setminus \mathcal{B}_i| < k_i$.

Zakleszczenie w modelu *OR-AND*

W modelu OR-AND zbiór warunkujący pasywnego procesu jest zdefiniowany jako $\mathcal{D}_i^1 \cup \mathcal{D}_i^2 \cup ... \cup \mathcal{D}_i^{qi}$, gdzie dla każdego naturalnego u, $1 \leqslant u \leqslant q_i$, $\mathcal{D}_i^u \subseteq \mathcal{P}$. Proces staje się aktywny po otrzymaniu wiadomości: od każdego z procesów tworzących zbiór $\mathcal{D}_i^1$, lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^2$, lub ... lub od każdego z procesów tworzących zbiór $\mathcal{D}_i^{qi}$.

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&(\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&(\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \wedge \\
&\qquad\qquad (\forall\ u :: 1 \leqslant u \leqslant q_i :: \\
&\qquad\qquad\quad (\exists\ P_j :: P_j \in \mathcal{D}_i^u \cap \mathcal{B} :: (\neg\ in\text{-}transit_i[j] \wedge \neg\ available_i[j]\ )))))
\end{aligned}
$$

Zakleszczenie w modelu *dysjunkcyjnym k spośród r*

W modelu *dysjunkcyjnym k spośród r* z każdym pasywnym procesem P_i skojarzony jest zbiór warunkujący $\mathcal{D}_i = \mathcal{D}_i^1 \cup \mathcal{D}_i^2 \cup ... \cup \mathcal{D}_i^{qi}$, liczby naturalne k_i^1, k_i^2, ..., k_i^{qi} i liczby naturalne r_i^1, r_i^2, ..., r_i^{qi}, gdzie $\mathcal{D}_i \subseteq \mathcal{P}$ oraz dla każdego naturalnego u, $1 \leqslant u \leqslant q_i$, $1 \leqslant k_i^u \leqslant r_i^u = |\mathcal{D}_i^u|$.

Proces staje się aktywny po otrzymaniu: wiadomości od co najmniej k_i^1 różnych procesów ze zbioru $\mathcal{D}_i^1$, lub wiadomości od co najmniej k_i^2 różnych procesów ze zbioru $\mathcal{D}_i^2$, lub ... lub wiadomości od co najmniej $k_i^{q_i}$ różnych procesów ze zbioru $\mathcal{D}_i^{q_i}$. Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&(\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&(\forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \wedge (\forall\ u :: 1 \leqslant u \leqslant q_i ::\\
&\qquad (\exists\ \mathcal{B}_i^u :: \mathcal{B}_i^u \subseteq \mathcal{D}_i^u \cap \mathcal{B} ::\\
&\qquad\quad (\ |\mathcal{D}_i^u \setminus \mathcal{B}_i^u| < k_i^u\ \ \wedge\\
&\qquad\qquad (\forall P_j :: P_j \in \mathcal{B}_i^u :: (\neg\ in\text{-}transit_i[j] \wedge \neg\ available_i[j]\ ))))))
\end{aligned}
$$

Zakleszczenie w modelu *predykatowym*

W modelu *predykatowym* dla każdego pasywnego procesu P_i ze zbiorem warunkującym $\mathcal{D}_i$ i dla każdego zbioru $\mathcal{X} \subseteq \mathcal{D}_i$ zdefiniowany jest predykat $activate_i(\mathcal{X})$. Predykat ten zachodzi, wtedy i tylko wtedy, gdy w związku z dostępnością wiadomości od wszystkich procesów tworzących zbiór, gotowe stanie się którekolwiek z dopuszczalnych zdarzeń odbioru. Wówczas:

$$
\begin{aligned}
&deadlock(\mathcal{B}) \equiv \\
&(\ \mathcal{B} \subseteq \mathcal{P}\ ) \wedge (\ \mathcal{B} \neq \emptyset) \wedge \\
&(\ \forall P_i :: P_i \in \mathcal{B} :: (\ passive_i \wedge \neg\ activate_i(\mathcal{AV}_i \cup \mathcal{IT}_i \cup (\mathcal{P} \setminus \mathcal{B}))))
\end{aligned}
$$

Definicja ta oznacza, że żaden proces $P_i \in \mathcal{B}$ nie może być uaktywniony nawet jeżeli uwzględnimy wszystkie wiadomości znajdujące się aktualnie w kanałach od oczekiwanych nadawców ($\mathcal{AV}_i \cup \mathcal{IT}_i$) oraz potencjalne wiadomości od wszystkich procesów nie zakleszczonych. Dotarcie wszystkich tych wiadomości nie jest bowiem wystarczające dla spełnienia warunku uaktywnienia któregokolwiek z procesów $P_i \in \mathcal{B}$.

Definicja zakleszczenia dla modelu predykatowego jest w pełni ogólna i może być dostosowana do innych, wcześniej omawianych modeli. Dlatego też ta definicja pozwala na jednorodne podejście do problemu zakleszczenia, abstrahując od konsekwencji szczególnych właściwości różnych modeli żądań.

Przy założeniu, że żaden proces nie osiąga stanu **zakończenia**, zakleszczenie dotyczy zawsze co najmniej dwóch procesów. Istotnie, przypuśćmy, że $\mathcal{B}=\{P_i\}$, zachodzi $deadlock(\mathcal{B})$ i żaden inny proces nie jest zakleszczony. Ponieważ $P_i \notin \mathcal{AV}_i \cup \mathcal{IT}_i$ otrzymujemy: $\mathcal{AV}_i \cup \mathcal{IT}_i \cup \mathcal{P} \setminus \mathcal{B} = \mathcal{P} \setminus \mathcal{B}$. Z drugiej strony, $\mathcal{D}_i \subseteq \mathcal{P} \setminus \mathcal{B} = \mathcal{P} \setminus \{P_i\}$ i stąd: $\neg activate_i(\mathcal{P} \setminus \mathcal{B}) \Rightarrow \neg activate_i(\mathcal{D}_i)$. Otrzymujemy zatem sprzeczność, gdyż z założenia $\mathcal{D}_i \neq \emptyset$, a z definicji predykatu $activate_i(\mathcal{X})$, $activate_i(\mathcal{D}_i) = True$.

Przykłady zakleszczeń

Dla ilustracji wprowadzonych definicji rozważmy przykład środowiska rozproszonego, składającego się z węzłów N_1, N_2, N_3, N_4 i N_5. Rozproszone przetwarzanie aplikacyjne obejmuje procesy P_1, P_2, P_3, P_4 i P_5 wykonywane w odpowiednich węzłach środowiska. Grafy na kolejnych slajdach przedstawiają tzw. grafy oczekiwanych potwierdzeń, oznaczane przez WFG (ang. *Wait-For-Graph*). Graf WFG jest grafem zorientowanym, którego wierzchołki reprezentują procesy, a łuki $\langle P_i, P_j \rangle$ reprezentują fakt, że proces P_i oczekuje na wiadomość od procesu P_j.

Oczywiście każdy proces w grafie WFG z łukiem wychodzącym jest pasywny. Procesy bez łuków wychodzących są natomiast aktywne. W rozważanym przykładzie załóżmy ponadto, że wszystkie kanały są puste.

Przykład – model jednostkowy

Pierwszy z grafów przedstawia zakleszczenie w modelu jednostkowym. Zbiory warunkujące są następujące: $\mathcal{D}_1=\{P_4\}$, $\mathcal{D}_2=\{P_4\}$, $\mathcal{D}_3=\{P_2\}$, $\mathcal{D}_4=\{P_3\}$. W analizowanym stanie zachodzi predykat $deadlock(\mathcal{B})$, dla $\mathcal{B}=\{P_1, P_2, P_3, P_4\}$.

Przykład – model AND

Następny graf ilustruje zakleszczenie w modelu AND. Mamy tu następujące zbiory warunkujące: $\mathcal{D}_1=\{P_4, P_5\}$, $\mathcal{D}_2=\{P_1, P_4\}$, $\mathcal{D}_3=\{P_2\}$, $\mathcal{D}_4=\{P_3\}$. W tym przykładzie $\mathcal{B}=\{P_1, P_2, P_3, P_4\}$ jest zbiorem zakleszczonych procesów.

Przykład – model OR

Kolejny z grafów przedstawia zakleszczenie w modelu OR. Zbiory warunkujące to: $\mathcal{D}_1=\{P_4, P_5\}$, $\mathcal{D}_2=\{P_4\}$, $\mathcal{D}_3=\{P_2\}$, $\mathcal{D}_4=\{P_2, P_3\}$. Predykat $deadlock(\mathcal{B})$ zachodzi tu dla $\mathcal{B}=\{P_2, P_3, P_4\}$.

Przyjmijmy teraz dla urozmaicenia, że w przetwarzaniu z następnego grafu kanały wejściowe procesów P_1, P_3, P_4 i P_5 są puste. W kanale $C_{3,2}$ jest natomiast dostępna wiadomość, ale proces P_2 nie oczekuje wiadomości od procesu P_3. Przyjmijmy ponadto, że wiadomość M wysłana z P_1 do P_2 jest właśnie transmitowana ($in\text{-}transit_2[1]=True$) lecz nie jest jeszcze dostępna. Ponieważ jednak $P_1 \notin \mathcal{D}_2$, nadejście wiadomości M, nie uaktywni procesu P_2. W rozważanym stanie, jedynie proces P_1 może być ewentualnie uaktywniony przez P_5. W konsekwencji, jeżeli modelem żądań jest model OR, to $deadlock(\mathcal{B}) = True$ dla $\mathcal{B}=\{P_2, P_3, P_4\}$. Zauważmy też jeszcze, że gdyby proces P_1 żądał jednocześnie wiadomości zarówno od P_5 jak i P_4 (jak to ma miejsce w modelu AND), to zbiór procesów zakleszczonych obejmowałby: P_1, P_2, P_3 i P_4.

Przykład – model k spośród r

Ostatni z grafów ilustruje zakleszczenie w modelu *k spośród r*. Zauważmy, że $\mathcal{D}_1=\{ P_2, P_4, P_5\}$, $\mathcal{D}_2=\{P_3\}$, $\mathcal{D}_3=\{P_2, P_4\}$, $\mathcal{D}_4=\{P_1, P_2, P_3\}$. Modele żądań dla P_1, P_2, P_3 i P_4 są zdefiniowane następująco: 1 *spośród* 3, 1 *spośród* 1, 2 *spośród* 2, oraz 2 *spośród* 3. W rozważanym stanie, proces P_1 może być uaktywniony, ale procesy P_2, P_3 i P_4 są zakleszczone.

Warto zauważyć, że predykat $deadlock(\mathcal{B})$ jest predykatem stabilnym. Tym samym zajście tego predykatu w pewnej chwili τ, implikuje, że dla każdego $\tau' \geqslant \tau$, $deadlock(\mathcal{B}) = True$. Powyższe stwierdzenie jest oczywiście poprawne, jeżeli rozważone są wyłącznie procesy tworzące określone przetwarzanie aplikacyjne. W szerszej jednak perspektywie, obejmującej dodatkowo procesy zewnętrzne w stosunku do aplikacji, jak na przykład monitory, możliwe staje się podjęcie działań zmierzających do przywrócenia stanu wolnego od zakleszczenia poprzez zewnętrzne wymuszenie zmiany stanu procesu aplikacyjnego (usunięcia procesu P_i i odzyskanie zasobów będących w jego dyspozycji). Takie podejście wymaga w pierwszej kolejności wykrycia (detekcji) zakleszczenia, co w środowisku rozproszonym nie jest jednak łatwe.

Klasyfikacja problemów detekcji zakleszczenia

Ogólnie rzecz ujmując, problem detekcji zakleszczenia przetwarzania rozproszonego sprowadza się do wyznaczenia (oceny) wartości predykatu $deadlock(\mathcal{B})$ przez grupę kooperujących monitorów. Monitory Q_i, podobnie jak procesy składowe P_i przetwarzania rozproszonego, działają w asynchronicznym środowisku rozproszonym. Właściwości tego środowiska stanowią podstawową przyczynę złożoności problemu detekcji zakleszczenia rozproszonego. Tak ogólnie sformułowany problem detekcji zakleszczenia można jednak uszczegółowić w różny sposób, definiując w efekcie problemy pochodne o różnym stopniu trudności, jak na przykład: problem detekcji wystąpienia zakleszczenia, problem detekcji zakleszczenia procesu, problem detekcji zakleszczenia zbioru procesów czy problem detekcji maksymalnego zbioru procesów zakleszczonych.

Detekcja wystąpienia zakleszczenia

Problem **detekcji wystąpienia zakleszczenia**, sprowadza się do znalezienia odpowiedzi na pytanie: Czy istnieje w pewnej chwili zbiór $\mathcal{B}$, dla którego predykat $deadlock(\mathcal{B})$ jest prawdziwy? Odpowiedź na to pytanie określa wartość predykatu:

$$dE \equiv (\exists \mathcal{B} :: deadlock(\mathcal{B}))$$

Detekcja wystąpienia zakleszczenia procesu

Detekcja zakleszczenia procesu P_i sprowadza się do sprawdzenia czy prawdziwy jest predykat:

$$dP_i \equiv (\exists \mathcal{B} :: deadlock(\mathcal{B}) \wedge P \in \mathcal{B}_i)$$

Detekcja wystąpienia zakleszczenia zbioru procesów

Detekcja zakleszczenia zbioru procesów, polega na znalezienia zbioru $\mathcal{B}$*, dla którego prawdziwy jest następujący predykat:

$$deadlock(\mathcal{B}^*) \vee ((\mathcal{B}^* = \emptyset) \wedge (\nexists \mathcal{B} :: deadlock(\mathcal{B})))$$

Detekcja maksymalnego zbioru zakleszczonego

Detekcja maksymalnego zbioru zakleszczonego, sprowadza się do znalezienia takiego zbioru $\mathcal{B}$*, dla którego spełniony jest warunek:

$$(deadlock(\mathcal{B}^*) \vee \mathcal{B}^* = \emptyset) \wedge (maxdead(\mathcal{B}^*)))$$

gdzie $maxdead(\mathcal{B}^*) \equiv (\forall \mathcal{B} :: deadlock(\mathcal{B}) \Rightarrow (\mathcal{B} \subseteq \mathcal{B}^*))$.

Model aplikacyjnego przetwarzania rozproszonego

W wielu zaproponowanych dotychczas algorytmach detekcji zakleszczenia przyjmuje się, że procesy aplikacyjne P_i wysyłają jawnie żądanie przydziału zasobów w formie wiadomości typu `REQUEST` do procesów tworzących zbiór warunkujący $\mathcal{D}_i$ i oczekują w stanie pasywnym na wiadomości typu `GRANT` potwierdzające przyznanie żądanych zasobów. Gdy do P_i dotrze odpowiedni zbiór wiadomości typu `GRANT`, proces aplikacyjny staje się aktywny i wysyła wiadomość typu `CANCEL`, do pozostałych procesów ze zbioru warunkującego (od których nie odebrał `GRANT`), w celu wycofania (unieważnienia) wcześniej wysłanego żądania. Po wykorzystaniu zasobu, proces aplikacyjny może sygnalizować jego zwolnienie przez wysłanie wiadomości typu `RELEASE` do procesu zarządzającego przydziałem zasobów.

Przy takim modelu aplikacyjnego przetwarzania rozproszonego stan globalny może być, jak już wspomniano, reprezentowany przez **graf oczekiwanych potwierdzeń WFG** (ang. *Wait-For-Graph*), w którym wierzchołki odpowiadają procesom P_i a łuki $\langle P_i, P_j \rangle$ reprezentują fakt, że

proces P_i wysłał już żądanie do P_j, lecz ani nie otrzymał jeszcze potwierdzenia GRANT, ani też nie wysłał wiadomości typu CANCEL. W tym kontekście zbiór warunkujący $\mathcal{D}_i$ jest więc zbiorem tych wszystkich procesów P_j, do których P_i wysłał żądanie REQUEST i ani nie zaniechał tego żądania (wysyłając następnie wiadomość CANCEL), ani też nie otrzymał potwierdzenia GRANT.

Algorytm Chandy, Misra, Hass dla modelu AND (1)

Znany jest algorytm detekcji zakleszczenia dla modelu żądań AND i środowiska z kanałami FIFO, w którym monitor pasywnego procesu inicjuje spontanicznie **przetwarzanie detekcyjne** (ang. *probe computation*).

W procesie detekcji monitory przesyłają wiadomości kontrolne typu PROBE. Wysłanie wiadomości kontrolnej może być realizowane jednocześnie przez wiele procesów i dlatego wiadomości te zawierają pole *initIndex*, określające indeks inicjatora.

Podstawowe zmienne wykorzystywane przez algorytm są następujące:

- *probeOut* – pakiet kontrolny
- $granted_i$ – tablica wartości logicznych, wartość *True* j-tego elementu tej tablicy oznacza że proces P_i po odebraniu ostatniej wiadomości REQUEST od procesu P_j wysłał do P_j potwierdzenie GRANT.
- $\mathcal{D}_i$ – zbiór warunkujący procesu P_i
- $recvProbe_i$ – tablica wartości logicznych, wartość *True* j-tego elementu tej tablicy oznacza że monitor Q_i odebrał od Q_j wiadomość typu PROBE zainicjowaną przez Q_k i spełnione są warunki konieczne zakleszczenia procesu P_i oraz P_j.
- α_i – indeks monitora, który zainicjował detekcję zakleszczenia
- $deadlockDetected_i$ – wartość True tej zmiennej oznacza że wykryte zostało zakleszczenie

Dla uproszczenia, pominięto w specyfikacji tego algorytmu oczywiste akcje monitora Q_i, związane ze zmianą wartości $granted_i$.

Algorytm Chandy, Misra, Hass dla modelu AND (2)

Detekcja zakleszczenia jest inicjowana przez monitor procesu pasywnego. Monitor taki zapamiętuje własny identyfikator w zmiennej α i rozsyła do pozostałych monitorów pakiet typu PROBE z własnym identyfikatorem informując je o rozpoczęciu detekcji zakleszczenia.

Algorytm Chandy, Misra, Hass dla modelu AND (3)

Jeżeli proces aplikacyjny staje się procesem aktywnym, czyli zachodzi zdarzenie *e_activate*, odpowiednie elementy tablicy *recvProbe* zostają ustawione na wartość *False*.

Algorytm Chandy, Misra, Hass dla modelu AND (4)

Monitor Q_i akceptuje wiadomość typu PROBE odebraną od Q_j i przesyła ją dalej do Q_k w wypadku, gdy spełnione są jednocześnie następujące warunki:

1. P_i jest pasywny,
2. P_i wysłał żądanie REQUEST i oczekuje na potwierdzenie GRANT od P_k, $(P_k \in \mathcal{D}_i)$
3. P_i nie wysłał potwierdzenia GRANT na ostatnie żądanie REQUEST od P_j,

4. otrzymana wiadomość PROBE jest pierwszą wiadomością tego typu od danego inicjatora Q_α, od momentu, gdy P_i zmienił swój stan na pasywny.

Jeżeli Q_α zaakceptuje wiadomość PROBE przez siebie zainicjowaną, to proces P_α jest zakleszczony.

Autorzy tego algorytmu wykazali, że po zainicjowaniu detekcji przez monitor Q_α procesu zakleszczonego, monitor ten stwierdzi w skończonym czasie, że P_α należy do cyklu w WFG.

Detekcja zakleszczenia dla modelu OR (1)

Algorytm detekcji zakleszczenia dla modelu *OR* opiera się w istocie na przetwarzaniu dyfuzyjnym nazywanym przez autorów **query computation**.

Właściwości algorytmu opisują następujące twierdzenia.

Detekcja zakleszczenia dla modelu OR (2)

Twierdzenie: Jeżeli inicjator Q_α rozpoczyna detekcję w chwili, gdy jego proces aplikacyjny P_α jest zakleszczony, to Q_α stwierdzi zakleszczenie procesu P_α w skończonym czasie (algorytm detekcji zakończy się).

Detekcja zakleszczenia dla modelu OR (3)

Twierdzenie: Jeżeli inicjator Q_α deklaruje, że jego proces aplikacyjny P_α jest zakleszczony, to P_α należy do pewnego zbioru procesów zakleszczonych w chwili zakończenia algorytmu.

Algorytm Chandy, Misra, Hass dla modelu OR (1)

Algorytm ten wykorzystuje dwa typy komunikatów kontrolnych, zapytania typu QUERY i odpowiedzi na zapytania typu REPLY. Każdy z tych komunikatów zawiera informację o procesie, który zainicjował dany proces detekcji oraz numer sekwencyjny przetwarzania sekwencyjnego.

Algorytm Chandy, Misra, Hass dla modelu OR (2)

Najistotniejsze zmienne wykorzystywane w przedstawionym algorytmie detekcji zakleszczenia dla modelu OR są następujące:

- $maxQueryNo_i[j]$ – oznacza największy numer sekwencyjny $queryNo$ spośród wszystkich zapytań QUERY zainicjowanych przez Q_j, a wysłanych lub odebranych przez Q_i.
- $engager_i[j]$ – indeks monitora Q_k, $k \neq i$, który spowodował zapisanie aktualnej wartości do $maxQueryNo_i[j]$.
- $QRBalance_i[j]$ – jest różnicą liczby zapytań QUERY zainicjowanych przez Q_j i wysłanych dalej przez Q_i oraz liczby odpowiedzi REPLY na te zapytania; wartość $QRBalance_i[j]=0$ oznacza, że Q_i otrzymał odpowiedzi na wszystkie zapytania związane z ostatnim procesem detekcji zainicjowanym przez Q_j.

- $contPassive_i[j]$ – jest $True$ wtedy i tylko wtedy, gdy P_i pozostawał pasywny przez cały czas od momentu ostatniego uaktualniania zmiennej $maxQueryNo_i[j]$; początkowo $contPassive_i[j]$ jest równe $False$.

Algorytm Chandy, Misra, Hass dla modelu OR (3)

Detekcja jest inicjowana przez jeden z monitorów Q_α, który wysyła do wszystkich monitorów procesów zbioru warunkującego $\mathcal{D}_\alpha$, wiadomość kontrolną `QUERY` zawierającą: indeks $initIndex$ inicjatora Q_α, oraz numer sekwencyjny $queryNo$ przetwarzania detekcyjnego zainicjowanego przez Q_α tym zapytaniem. W odpowiedzi na zapytanie `QUERY`, monitory Q_i oczekują odpowiedzi `REPLY` zawierającej takie same wartości $initIndex$ i $queryNo$.

Algorytm Chandy, Misra, Hass dla modelu OR (4)

W przypadku odebrania przez monitor Q_i wiadomości typu `QUERY` od monitora Q_j monitor ten sprawdza czy monitorowany przez niego proces jest pasywny. Jeśli warunek ten jest spełniony to sprawdzany jest numer sekwencyjny zapytania `QUERY`. Jeśli jest on większy od największego otrzymanego dotychczas to aktualizowane są odpowiednio tablice $maxQueryNo_i$, $engager_i$, i $contPassive_i$. Następnie przygotowywany jest pakiet zawierający informację o tym numerze sekwencyjnym i inicjatorze detekcji, który zostanie wysłany do wszystkich monitorów, których monitorowane procesy należą do zbioru warunkującego procesu P_i. Po wysłaniu tej wiadomości tablicy $QRBalance_i$ na pozycji odpowiadającej identyfikatorowi przypisywana jest wartość równa liczbie procesów tworzących zbiór warunkujący.

Algorytm Chandy, Misra, Hass dla modelu OR (5)

Jeśli natomiast wartość odebranego w pakiecie numeru sekwencyjnego jest równa wartości zapamiętanej w tablicy $maxQueryNo_i$ i proces był cały czas pasywny od czasu ostatniej aktualizacji porównywanego elementu tej tablicy, to przygotowywana jest odpowiedź typu `REPLY`, która następnie jest wysyłana do procesu Q_j.

Algorytm Chandy, Misra, Hass dla modelu OR (6)

Zdarzenie odbioru wiadomości typu `REPLY` powoduje w przypadku gdy proces jest pasywny porównanie numerów sekwencyjnych zapisanych w odebranej wiadomości i odpowiedniej tablicy $maxQueryNo_i$. Jeśli okaże się że wartości te są sobie równe, a ponadto proces aplikacyjny P_i jest procesem pasywnym od czasu ostatniej aktualizacji porównywanego elementu tablicy $maxQueryNo_i$ to zmniejszana o 1 na odpowiedniej pozycji. jest wartość zmiennej $QRBalance_i$. Jeśli po zmniejszeniu wartość tej zmiennej wynosi 0 i monitor odbierający komunikat jest inicjatorem detekcji to stwierdza on, że proces aplikacyjny, który monitoruje jest zakleszczony.

Algorytm Chandy, Misra, Hass dla modelu OR (7)

Jeśli natomiast wartość zmiennej $QRBalance_i$ po aktualizacji jest równa 0, czyli proces otrzymał już wszystkie odpowiedzi na zapytania związane z ostatnim procesem detekcji zainicjowanym przez proces o indeksie α_i, ale proces odbierający wiadomość nie jest inicjatorem detekcji to wysyłana jest wiadomość typu `REPLY` do procesu, od którego proces ten otrzymał wiadomość typu `QUERY`, w wyniku której nastąpiła aktualizacja tablicy $maxQueryNo_i$ na odpowiedniej pozycji.

Algorytm Chandy, Misra, Hass dla modelu OR (8)

Uaktywnienie procesu P_i powoduje ustawienie elementów tablicy $contPassive_i$ odpowiadających procesowi P_i na *False*.

Algorytm Bracha, Toueg'a (1)

Bracha i Toueg przedstawili algorytm detekcji zakleszczenia rozproszonego dla modelu żądań *k spośród r* przyjmując, że komunikacja jest natychmiastowa. Dla tego przypadku w każdym obrazie spójnym przetwarzania rozproszonego wszystkie kanały są puste. Obraz spójny wyrażono następnie w formie grafu oczekiwanych potwierdzeń $WFG = \langle \mathcal{P}, \mathcal{A} \rangle$, w którym wierzchołki odpowiadają procesom P_i a łuki $\langle P_i, P_j \rangle$, oznaczone przez $A_{i,j}$ reprezentują fakt, że proces P_i wysłał już żądanie `REQUEST` do P_j, lecz nie otrzymał jeszcze potwierdzenia `GRANT`, ani też nie wysłał unieważnienia `CANCEL`.

W tym kontekście, zbiór $\mathcal{OUT}_i = \{P_j : \langle P_i, P_j \rangle \in \mathcal{A}\}$ jest więc zbiorem warunkującym $\mathcal{D}_i$ procesu P_i.

Niech ponadto $\mathcal{IN}_i$ będzie zbiorem procesów P_j, od których P_i odebrał wiadomość `REQUEST`, lecz ani nie wysłał jeszcze w odpowiedzi potwierdzenia `GRANT`, ani też nie otrzymał od P_j unieważnienia `CANCEL`.

Zauważmy teraz, że w środowisku z komunikacją natychmiastową, każda wiadomość `REQUEST`, `GRANT` czy `CANCEL` jest od razu odebrana, a więc $\mathcal{IN}_i = \{P_j: \langle P_j, P_i \rangle \in \mathcal{A}\}$, a ponadto $P_i \in \mathcal{OUT}_j$ wtedy i tylko wtedy, gdy $P_j \in \mathcal{IN}_i$.

Przedstawimy teraz algorytm detekcji zakleszczenia dla danego grafu oczekiwanych potwierdzeń $WFG = \langle \mathcal{P}, \mathcal{A} \rangle$. Dla uproszczenia prezentacji założymy, że w danym czasie wykonywany jest co najwyżej jeden proces detekcji i taki właśnie pojedynczy proces będziemy dalej rozważać. Algorytm detekcji składa się z dwóch faz: powiadamiania i potwierdzania. W fazie **powiadamiania** (ang. *notify*) wszystkie monitory są informowane o rozpoczęciu detekcji, natomiast w fazie **potwierdzania** (ang. *confirm*) monitor każdego procesu aktywnego, lub potencjalnie aktywnego, symuluje wysłanie potwierdzenia typu `GRANT`.

Wykorzystywane są cztery typy komunikatów będących sygnałami. Są to typy: `NOTIFY`, `DONE`, `CONFIRM` i `ACK`.

Algorytm Bracha, Toueg'a (2)

Najważniejsze zmienne wykorzystywane przez ten algorytm są następujące:

- $\mathcal{A}$ – zadany zbiór łuków grafu WFG
- $\mathcal{OUT}_i$ – zbiór procesów P_j, do których P_i wysłał żądanie `REQUEST` i nie otrzymał jeszcze potwierdzenia GRANT, ani nie wysłał unieważnienia `CANCEL`
- $\mathcal{IN}_i$ – zadany zbiór łuków wejściowych wierzchołka P_i grafu WFG
- $expectNo_i$ – żądana przez proces aplikacyjny liczba oczekiwanych wiadomości

Algorytm Bracha, Toueg'a (3)

Procedura `NotifyProc` polega na rozesłaniu do wszystkich procesów należących do zbioru procesów sąsiednich wyjściowych, czyli należących do zbioru warunkującego procesu P_i wiadomości typu `NOTIFY` i zebraniu potwierdzeń typu `DONE`. Ilustruje ona pierwszą fazę algorytmu (fazę powiadamiania), w której wszystkie monitory są informowane o rozpoczęciu detekcji. Należy zauważyć że wywołanie procedury `ConfirmProc`, realizującej drugą fazę

algorytmu jest zagnieżdżone w fazie powiadamiania, która tym samym kończy się dopiero po zakończeniu fazy drugiej.

Algorytm Bracha, Toueg'a (4)

Procedura CONFIRMPROC jak już wcześniej wspomniano jest odpowiedzialna za realizację drugiej fazy przetwarzania, w której monitor Q_i rozsyła do wszystkich procesów ze zbioru sąsiednich procesów wejściowych procesu P_i wiadomość typu CONFIRM i czeka na otrzymanie od wszystkich potwierdzenia otrzymania tej wiadomości, czyli komunikatu typu ACK.

Algorytm Bracha, Toueg'a (5)

Algorytm rozpoczyna inicjator Q_α, którego proces aplikacyjny jest pasywny, a więc potencjalnie zakleszczony wywołując procedurę NOTIFYPROC.

Algorytm kończy się, gdy dalsze uaktywnienia nie są już możliwe i w związku z tym żadne dodatkowe wiadomości typu CONFIRM nie zostaną przesłane. Jeśli po zakończeniu algorytmu, dla pewnych procesów w dalszym ciągu nie są potencjalnie spełnione warunki uaktywnienia, to procesy te są uznane za zakleszczone.

Algorytm Bracha, Toueg'a (6)

Przesyłanie wiadomości typu CONFIRM między monitorami pozwala stwierdzić, czy uaktywnienie procesu jest możliwe. Jeśli tak, to odpowiedni monitor symuluje możliwe zachowanie uaktywnionego procesu i przesyła do kolejnych monitorów wiadomości typu CONFIRM, oczekując ich potwierdzenia.

Algorytm Bracha, Toueg'a (7)

Odebranie wiadomości typu NOTIFY przez proces, który jeszcze nie otrzymał tego typu wiadomości, powoduje że proces ten rozsyła tą wiadomość dalej wywołując procedurę NOTIFYPROC.

Przykład działania algorytmu (1)

Dla ilustracji działania algorytmu rozważmy przykładowy stan globalny reprezentowany przez graf oczekiwanych potwierdzeń przedstawiony na slajdzie. Procesy P_1, P_2, P_3 i P_4 należące do zbioru $\mathcal{P}$ są pasywne i oczekują na wiadomości od innych procesów ze zbioru $\mathcal{P}$, zgodnie z grafem WFG. Dla prostoty prezentacji skojarzymy z każdym wierzchołkiem grafu WFG (procesem) dwa wektory reprezentujące stan algorytmu w danym wierzchołku. Pierwszy wektor, oznaczony przez pS_i (ang. *process state*), jest zmienną reprezentowaną przez tablicę [1..2] liczb naturalnych. Wartość $pS_i[1]$ zależy od wartości zmiennej $notified_i$, i od liczby odebranych wiadomości typu DONE. Początkowo wartość $pS_i[1]$ jest równa 0. Po nadaniu zmiennej $notified_i$ wartości $True$ przez monitor Q_i, $pS_i[1]$ przyjmuje wartość 1. Następnie, gdy zakończy się zainicjowana ewentualnie faza potwierdzania i odebrana zostanie ostatnia wiadomość typu DONE, wartość $pS_i[1]$ jest zmieniana na 2. Podobnie wartość $pS_i[2]$ zależy od zmiennej $confirmed_i$, oraz od liczby odebranych wiadomości typu ACK. Początkowo $pS_i[2]$ równa się 0. Po nadaniu zmiennej $confirmed_i$ wartości $True$ przez monitor Q_i, $pS_i[2]$ przyjmuje wartość 1. Następnie, po odebraniu ostatniej wiadomości typu ACK, wartość elementu $pS_i[2]$ jest zmieniana na 2. Drugi wektor skojarzony z każdym monitorem Q_i, oznaczony przez cS_i (ang. *communication state*), jest zmienną reprezentowaną przez tablicę [1..3], której elementy odpowiadają licznikom odebranych wiadomości typu CONFIRM, ACK i DONE. Początkowo, wszystkie te liczniki mają wartość 0. Są one stosownie zwiększane w wyniku odebrania

wiadomości typu CONFIRM, ACK i DONE. Na rysunku wektor pS_i jest umieszczony wewnątrz wierzchołka, a wektor cS_i obok wierzchołka.

W rozważanym przykładzie przyjęto, że $Q_1 = Q_\alpha$ jest jedynym inicjatorem procesu detekcji zakleszczenia. Monitor Q_1 wywołuje procedurę NOTIFYPROC i w konsekwencji jego stan zmienia się, wiadomości typu NOTIFY zostają wysłane (co zaznaczono białym kółkiem i strzałką) do monitorów wszystkich procesów tworzących zbiór warunkujący $\mathcal{OUT}_1$, a monitor oczekuje na wiadomości typu DONE potwierdzające wszystkie wiadomości typu NOTIFY. Następnie, monitory Q_2, Q_3 i Q_4 odbierają współbieżnie wiadomości typu NOTIFY i w konsekwencji wywołują procedurę NOTIFYPROC, zmieniając tym samym odpowiednio $pS_i[1]$ i wysyłając wiadomości typu NOTIFY do monitorów wszystkich procesów tworzących ich zbiory warunkujące. W następnym kroku, wiadomości typu NOTIFY docierają do monitorów, których flagi $notified_i$ są już zapalone ($True$). Dlatego, monitory te wysyłają w odpowiedzi wiadomości typu DONE (zaznaczone czarnym kółkiem). W rezultacie, wektory stanów monitorów Q_2, Q_3 i Q_4 przyjmują wartość [2,0]. Umożliwia to dalej wysłanie wiadomości typu DONE do inicjatora Q_1. Gdy inicjator otrzyma wszystkie oczekiwane wiadomości typu DONE, algorytm kończy się stwierdzając zakleszczenie procesu P_1.

Przykład działania algorytmu (2)

Drugi przykład zastosowania algorytmu przedstawia kolejny slajd. Rozważany teraz graf jest podobny do poprzedniego, z tą tylko różnicą, że węzeł P_4 nie ma krawędzi wychodzącej. Oznacza to, że proces P_4 jest aktywny. Załóżmy jeszcze, że każdy proces oczekuje potwierdzeń od wszystkich procesów tworzących jego zbiór warunkujący. Niech ponadto Q_1 będzie ponownie jedynym inicjatorem. Rozpoczyna on fazę powiadamiania, wysyłając wiadomość typu NOTIFY do monitorów wszystkich procesów tworzących zbiór $\mathcal{OUT}_i$. Kiedy wiadomość typu NOTIFY dotrze do Q_4, którego $\mathcal{OUT}_4 = \emptyset$, monitor Q_4 wywołuje procedurę NOTIFYPROC. Procedura ta zmienia stan $pS_4[1]$ na 1, a następnie wywołuje procedurę CONFIRMPROC. W efekcie, $pS_4[1] = 1$ i wiadomość typu CONFIRM zostaje wysłana do monitorów wszystkich procesów tworzących zbiór $\mathcal{IN}_4$. W kroku tym, monitory Q_2 i Q_3 zmieniają swoje zmienne stanu $pS_2[1]$ oraz $pS_3[1]$ na 1 i wysyłają wiadomości typu NOTIFY do monitorów wszystkich procesów tworzących ich zbiory $\mathcal{OUT}_i$. Tak więc monitor Q_4 jest inicjatorem fazy potwierdzania dla Q_1, Q_2 i Q_3. Tym samym Q_4 wyśle wiadomość typu DONE do inicjatora jego fazy powiadamiania, czyli do Q_1, dopiero po otrzymaniu wiadomości typu ACK, w odpowiedzi na wszystkie wiadomości typu CONFIRM. W następnym kroku, między innymi monitor Q_3 otrzymuje wiadomość typu CONFIRM od Q_4 i przekazuje ją do wszystkich swoich poprzedników w grafie.

Należy zauważyć, że w kroku tym Q_2 nie przesyła wiadomości typu CONFIRM do Q_1, gdyż w przyjętym modelu żądań jest to dopuszczalne dopiero po otrzymaniu wiadomości typu CONFIRM od wszystkich następników w grafie. Warunek ten jest spełniony w następnym kroku, kiedy to Q_2 otrzymał już wiadomość typu CONFIRM od Q_3. Wówczas, Q_2 wysyła wiadomość typu CONFIRM do Q_1. W efekcie, w następnym kroku wiadomości typu ACK przesłane są kolejno do Q_2, Q_3 i Q_4. Gdy wiadomość typu ACK dotrze do Q_4, monitor ten kończy fazę potwierdzania, co umożliwia mu wysłanie wiadomości typu DONE do monitora będącego dla Q_4 inicjatorem jego fazy powiadamiania.

Złożoność czasowa algorytmu detekcji zakleszczenia w środowisku synchronicznym dla modelu k spośród r (1)

Przeanalizujmy teraz złożoność czasową tego algorytmu, przyjmując, że graf niezorientowany odpowiadający grafowi zorientowanemu, scharakteryzowany jest przez średnicę grafu *d* i najdłuższą ścieżkę w grafie – *l*. W ostatnim z prezentowanych przykładów, $d = 1$ a $l = 3$.

Ponieważ faza potwierdzania jest zagnieżdżona w fazie powiadamiania, to aby wyznaczyć złożoność całego algorytmu trzeba dodać do złożoności fazy potwierdzania złożoność powiadomienia inicjatora fazy potwierdzania (przekazania mu wiadomości typu NOTIFY), jak również złożoność odpowiadającą przesłaniu wiadomości typu DONE od inicjatora fazy potwierdzania do inicjatora detekcji.

Złożoność czasowa algorytmu detekcji zakleszczenia w środowisku synchronicznym dla modelu k spośród r (2)

Zauważmy najpierw, że faza potwierdzania jest propagowana (wiadomość typu CONFIRM zostaje wysłana do następników) przez monitor procesu pasywnego, dopiero po otrzymaniu odpowiedniej liczby wiadomości typu CONFIRM. W ogólności oznacza to, że muszą dotrzeć wiadomości typu CONFIRM od wszystkich następników (tak jak w naszym przykładzie). Czas wymagany do osiągnięcia takiego stanu detekcji zależy od najdłuższej ścieżki między inicjatorem fazy potwierdzania a danym monitorem. Dla uzasadnienia tego stwierdzenia rozważmy graf, w którym inicjator Q_x fazy potwierdzania jest połączony ze swoim poprzednikiem Q_y dwoma ścieżkami o długości l_1 i l_2, gdzie $l_1 < l_2$. W tym wypadku, pierwsza wiadomość typu CONFIRM może dotrzeć do Q_y po l_1 jednostkach czasu.

Przypuśćmy teraz, że do uaktywnienia wymagane są dwie wiadomości typu CONFIRM. Ta druga wiadomość typu CONFIRM wymaga jednak $l_2 > l_1$ jednostek czasu i stąd obie wiadomości stają się dostępne po czasie równym $max(l_1, l_2)$. W ogólności jednak, l_2 może być równe l – długości najdłuższej ścieżki w grafie. Tak więc, nawet gdy pominiemy opóźnienia wnoszone przez węzły, przesłanie wiadomości typu CONFIRM może zabrać l jednostek czasu. Następnie jednak, wiadomości typu ACK przesłane są w kierunku przeciwnym do odpowiednich inicjatorów fazy potwierdzania i ich transmisja również zabiera w najgorszym wypadku l jednostek czasu.

Z drugiej strony zauważmy też, że wiadomość typu ACK jest wysłana natychmiast po otrzymaniu wiadomości typu CONFIRM, jeśli monitor otrzymał już wcześniej wiadomość typu CONFIRM, lub gdy warunek uaktywnienia nie jest spełniony. Stąd w ogólności, pesymistyczna złożoność czasowa fazy potwierdzania wynosi $2l$.

Ponieważ wiadomości typu NOTIFY są przysyłane równolegle do monitorów wszystkich procesów tworzących zbiór $\mathcal{OUT}_i$, więc przesłanie to wymaga co najwyżej d kroków (jednostek czasu). To samo można stwierdzić o czasie propagacji wiadomości typu po zakończeniu fazy potwierdzania. W efekcie dochodzimy do wniosku, że złożoność czasowa algorytmu wynosi $2d + 2l$. Wniosek ten potwierdza ostatnio prezentowany przykład. Rzeczywiście, ponieważ analizowany graf jest scharakteryzowany przez średnicę d=1 oraz l=3, więc liczba kroków w najgorszym przypadku powinna wynosić 8. Ta sama liczba kroków okazała się również niezbędna w naszym przykładzie.