Materiały do wykładów z analizy algorytmów

Algorytmy samostabilizujące

Jakub Lemiesz

1 Samostabilizacja

Samostabilizacja to idea mająca na celu uodpornienie algorytmów działających w systemach rozproszonych na błędy. W dużym skrócie idea ta mówi, że jeśli w wyniku błędu algorytm samostabilizujący znajdzie się w stanie niepoprawnym, to w skończonej liczbie kroków samoistnie przejdzie do stanu poprawnego. Ponieważ dopuszczamy pojawienie się stanu niepoprawnego i jego tymczasowe utrzymywanie się w algorytmie, uzyskane gwarancje mogą wydawać się nieco słabsze od klasycznych, niedopuszczających możliwości zaistnienia takiego stanu. Zauważmy jednak, że całkowite wykluczenie pojawienia się stanu niepoprawnego nie jest możliwe. W szczególności algorytm może zostać uruchomiony w stanie niepoprawnym, mogą nastąpić losowe lub wprowadzone przez adwersarza błędy pamięci, mogą pojawić się zmiany w składzie lub topologii sieci. Rozważenie wszystkich scenariuszy dla algorytmów działających w środowisku rozproszonym często bywa dużym wyzwaniem, m.in. ze względu na ich złożoność oraz trudności z debugowaniem. Samostabilizacji pozwala zniwelować tego rodzaju problemy, również takie, których nie przewidziano przy projektowaniu algorytmu.

Idea samostabilizacji pochodzi z pracy Self-stabilizing systems in spite of distributed control Edsgera Dijkstry z 1974 r. w której analizowane są samostabilizujące się algorytmy wzajemnego wykluczenia. Idea została dostrzeżona i doceniona pod koniec lat 90-tych. W 2002 r. powyższy artykuł Dijkstry [1] z 1974 r. otrzymał jedną z ważniejszych nagród w społeczności związanej z przetwarzaniem rozproszonym i przyznawanej z istotny i wieloletni wpływ zarówno na prowadzone badania jak i rozwiązania praktyczne ACM PODC Influential-Paper Award. Po śmierci Dijkstry nazwę nagrody zmieniona na Dijkstra Prize.

Teoria związana z algorytmami samostabilzującymi i analiza przykładowych algorytmów znajdują się w m.in. w książkach:

1. Self-Stabilization, Shlomi Dolev, MIT Press, 2000, [2],
2. Introduction to Distributed Algorithms, Gerard Tel, Cambridge University Press, 2001, [3].

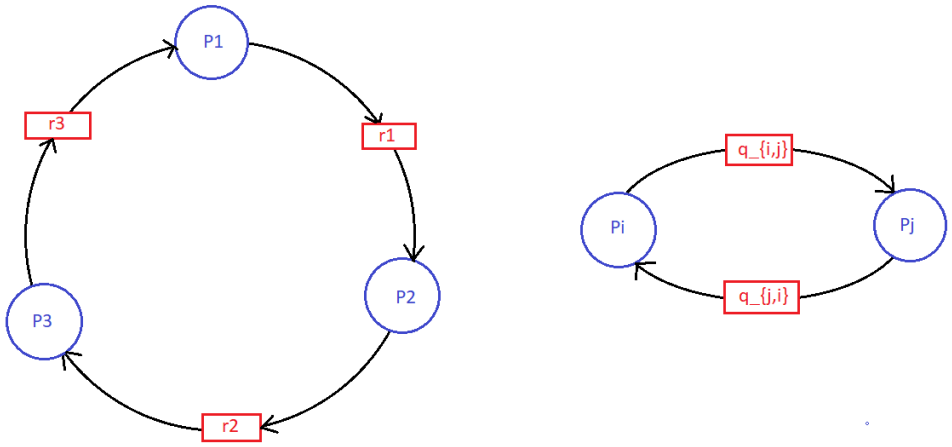
Przedstawiony dalej zarys teorii jest próbą kompresji (bardzo stratną) informacji o samostabilizacji zawartych w powyższych książkach.

2 Model

Poniżej przedstawiamy w bardzo skróconej formie opis modelu, w ramach którego będziemy projektować i analizować algorytmy.

1. Rozważamy system rozproszony składający się z niezależnych procesów $P_1, P_2, \ldots, P_n$.
2. To które pary procesów mogą się komunikować określa graf komunikacji G, w którym wierzchołki odpowiadają procesom a skierowane krawędzie możliwości przesłania komunikatu.

3. Komunikację między dwoma procesami modeluje się za pomocą współdzielonych rejestrów lub kolejek FIFO kolejkujących wiadomości. O rejestrach myślimy najczęściej gdy mamy do czynienia z procesami działającymi w ramach jednej maszyny. Typowo proces P_i ma wyłączne prawo pisania do rejestru r_i, a jego sąsiedzi w grafie G mogą jedynie czytać z r_i. Kolejka $q_{i,j}$ służy do modelowania komunikacji między procesem P_i a procesem P_j.



Rysunek 1: Przykładowe grafy komunikacji:

(a) z lewej graf o topologii pierścienia: proces P_i może pisać do rejestru r_i, a jego następca w pierścieniu czytać z tego rejestru,

(b) z prawej modelowanie komunikacji za pomocą kolejek. Kolejka $q_{i,j}$ odpowiada komunikatom przesyłanym od P_i do P_j.

4. Konfiguracją systemu nazywamy stan wszystkich zmiennych, a także rejestrów lub kolejek w danej chwili. Konfigurację możemy zapisać np. jako $c = (s_1, s_2, \ldots, s_n, r_1, r_2, \ldots, r_n)$, gdzie s_i oznacza stan i-tego procesu (wartości wszystkich wykorzystywanych zmiennych) a r_i wartość i-tego rejestru.

5. Wykonaniem (ang. execution) nazywamy sekwencję konfiguracji, np. $e = (c_1, c_2, c_3, \ldots)$.

6. Rozważa się modele asynchroniczne i synchroniczne, z praktycznego punktu widzenia interesujące są przede wszystkim te pierwsze. Asynchroniczność modelu oznacza m.in. że procesy mogą wykonywać inną liczbę kroków algorytmu w tym samym czasie, a czas przesyłania komunikatów jest niedeterministyczny. Asynchroniczność oznacza w szczególności, że algorytm uruchomiony dwa razy w tej samej konfiguracji może mieć inne wykonania.

   W analizie algorytmów samostabilizujących często pojawia się Central Deamon - przeciwnik mający uniemożliwić stabilizację wskazując, który proces ma wykonać następny krok. Ograniczaniem przeciwnika jest to, że nie może zagłodzić żadnego procesu, tzn. odwlekać jego ruchu w nieskończoność (ang. fair execution).

   Rundą określa się cześć wykonania, w której każdy proces miał możliwość przejścia co najmniej jednego kroku algorytmu. Czas potrzebny na stabilizację algorytmu często opisuje się przez liczbę potrzebnych rund.

7. Specyfikacja S opisuje pożądany stan algorytmu. Specyfikację typowo określa się na ciągu konfiguracji, np. „w każdej konfiguracji tylko jeden proces może być w sekcji krytycznej”.

3 Definicja samostabilizacji

Definicja 1 (Samostabilizacja) *Niech C oznacza zbiór możliwych konfiguracji, $P \subseteq C$ oznacza zbiór poprawnych konfiguracji, a S specyfikację. Mówimy, że algorytm samostabilizuje się do specyfikacji S, gdy jest*

a) poprawny: każde wykonanie rozpoczynające się w konfiguracji $c \in P$ spełnia specyfikację S,

b) zbieżny: każde wykonanie osiąga pewną konfigurację $c \in P$.

Lemat 1 *Jeśli system samostabilizuje się do specyfikacji S to każde wykonanie $e = (c_1, c_2, \ldots)$ ma sufiks spełniający warunki S.*

Dowód. Zauważmy, że

a) każde wykonanie osiąga pewną konfigurację $c \in P$ (zbieżność),

b) sufiks rozpoczynający się od konfiguracji c spełnia warunki S (poprawność).

■

Istnieją różne technik dowodzenia samostabilizacji. Standardowo dowodzi się poprawności pokazując, że algorytm sam z siebie nie powoduje przejścia z poprawnej do niepoprawnej konfiguracji, a następnie zbieżności konstruując funkcję potencjału $f : C \to D$ odwzorowującą zbiór konfiguracji C w pewien dobrze uporządkowany zbiór D. Funkcja f powinna zapewniać, że dla każdego kroku algorytmu powodującego przejście z konfiguracji c_i do c_j zachodzi jeden z dwóch warunków $f(c_i) > f(c_j)$ lub $c_j \in P$. Ponieważ zbiór D jest dobrze uporządkowany, a wartości f maleją w kolejnych przejściach, poprawna konfiguracja będzie osiągnięta w skończonej liczbie kroków. Ponadto, jeśli c_i jest konfiguracją ostateczną, tzn. żaden krok algorytmu nie może z niej wyprowadzić, to musi zachodzić warunek $c_i \in P$.

4 Problem maksymalnego sparowania

Przedstawimy teraz algorytm samostabilizujący dla problemu maksymalnego sparowania w grafie (ang. maximal matching). Zacznijmy od zdefiniowania problemu.

Definicja 2 (Maksymalne sparowanie) *Niech $G = (V, E)$ będzie grafem nieskierowanym, tzn.*

$$E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V, i \neq j\} \ .$$

Maksymalnym sparowaniem nazywamy podzbiór $M \subseteq E$ taki, że

a) każdy $v \in V$ może być w co najwyżej jednej parze $\{v, u\} \in M$ (warunek sparowania),

b) M nie można powiększyć przez dodanie nowych krawędzi (warunek maksymalności).

Zauważmy, po pierwsze, że interesuje nas dowolne maksymalne sparowanie, a nie największe możliwe. Po drugie zauważmy, że problem ten można łatwo rozwiązać algorytmem off-line (tzn. jeśli cały graf jest dany) np. stosując strategię zachłanną. Nas jednak interesuje rozwiązanie problemu w systemie rozproszonym, w którym żaden wierzchołek nie ma pełnej wiedzy, a jedynie możliwość komunikacji z sąsiadami w grafie. Ponadto chcielibyśmy zaprojektować algorytm samostabilizujący, a w szczególności taki, który samoistnie radziłby sobie ze zmianami w grafie komunikacji, np. pojawianie się i znikanie wierzchołków, zmiany topologii. Zastosowaniem takiego rozwiązania może być np. stabilny podział na klastry obliczeniowe.

5 Algorytm samostabilizujący dla maksymalnego sparowania

Przyjmijmy, że każdy proces p kontroluje jeden rejestr przechowujący informację o jego aktualnej preferencji $pref_p \in N(p) \cup \{NULL\}$, gdzie $N(p)$ oznacza zbiór sąsiadów p w grafie komunikacji G. Wartość $NULL$ oznacza, że proces aktualnie nie ma preferencji. Zauważmy, że wszystkie możliwe stany procesu wyrażają się za pomocą następujących predykatów:

1) $married(p) \equiv pref_p = q \in N(p) \wedge pref_q = p \in N(q)$,

2) $single(p) \equiv pref_p = NULL \wedge \left(\forall_{q \in N(p)}\right) (married(q))$,

3) $free(p) \equiv pref_p = NULL \wedge \left(\exists_{q \in N(p)}\right) (\neg married(q))$,

4) $wait(p) \equiv pref_p = q \in N(p) \wedge pref_q = NULL$,

5) $chain(p) \equiv pref_p = q \in N(p) \wedge pref_q = r \in N(q) \wedge r \neq p$.

Przyjmujemy, że każde wykonanie jednego przebiegu pętli poniższego algorytmu przez proces p to niepodzielny krok. Istnieją wersje algorytmu bez tego założenia. Wiąże się to z faktem, że algorytmy samostabilizacyjne można ze sobą składać zachowując własność samostabilizacji. W tym przypadku można by np. wykorzystać samostabilizujący algorytm wzajemnego wykluczania.

Każdy proces p wykonuje następującą pętle

do forever

1: **if** $pref_p = NULL \wedge \left(\exists_{q \in N(p)}\right) (pref_q = p)$ **then** ▷ accept proposal
2: $\quad pref_p \leftarrow q$
3: **end if**
4: **if** $pref_p = NULL \wedge \left(\forall_{q \in N(p)}\right) (pref_q \neq p) \wedge \left(\exists_{q \in N(p)}\right) (pref_q = NULL)$ **then** ▷ propose
5: $\quad pref_p \leftarrow q$
6: **end if**
7: **if** $pref_p = q \wedge pref_q \neq p \wedge pref_q \neq NULL$ **then** ▷ unchain
8: $\quad pref_p \leftarrow NULL$
9: **end if**

Specyfikację S definiujemy następująco: w każdej poprawnej konfiguracji prawdziwe jest zdanie

$$(\forall p) (married(p) \vee single(p)) .$$

Aby dowieść, że algorytm samostabilizuje się do specyfikacji S, która odpowiada maksymalnemu sparowaniu należy udowodnić następujące lematy. Lematy spróbuj udowodnić samodzielnie, w razie problemów możesz zajrzeć do książki [3] na stronę 530 i dalej.

Lemat 2 *Jeśli zachodzą warunki specyfikacji S, to zbiór $M = \{\{p, pref_p\} : pref_p \neq NULL\}$ jest maksymalnym sparowaniem.*

Dowód. Wskazówka: pokaż nie wprost, że M jest poprawnym sparowaniem (tzn. że każdy proces p jest w co najwyżej jednej parze). Następnie pokaż nie wprost, że M jest sparowaniem maksymalnym. ■

Lemat 3 (Poprawność z Definicji 1) *Zachodzą warunki specyfikacji $S \iff$ konfiguracja jest ostateczna (tzn. żaden krok algorytmu nie może zmienić konfiguracji).*

Dowód. Wskazówka: ($\Rightarrow$) pokaż, że jeśli zachodzą warunki S to żadna akacja algorytmu nie jest możliwa. ($\Leftarrow$) d-d nie wprost: pokaż, że jeśli nie zachodzą warunki specyfikacji S to konfiguracja może się zmienić. ■

Lemat 4 (Zbieżność z Definicji 1) *Jeśli mamy n procesów to algorytm osiąga konfigurację legalną w nie więcej niż $\mathcal{O}\left(n^3\right)$ krokach.*

Dowód. Wskazówka: rozważ funkcję potencjału $f(t) = (c, f, w, m+s)$, gdzie c to liczba procesów w stanie „chain", f to liczba procesów w stanie „free", w to liczba procesów w stanie „wait", $m+s$ to liczba procesów w stanie „married" oraz „single" w kroku t od konfiguracji początkowej. Zauważ, że porządek leksykograficzny na krotkach jest dobrym porządkiem. ■

Literatura

[1] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[2] S. Dolev. *Self-Stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[3] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, USA, 2nd edition, 2001.