Algorytmy Równoległe i Rozproszone
Część VII - Systemy rozproszone, c. d.

Łukasz Kuszner
pokój 209, WETI
http://www.sphere.pl/~kuszner/
kuszner@sphere.pl

Oficjalna strona wykładu
http://www.sphere.pl/~kuszner/ARiR/
Wykład 15 godzin, Projekt 15 godzin

2007/08

1. Model synchroniczny all-port

W modelu *all-port* zakładamy:

1. gwarantowaną niezawodność obliczeń i komunikacji,
2. jednostkowy czas przesłania i odebrania komunikatów do wszystkich wierzchołków sąsiednich w grafie systemu,
3. pomijalnie mały czas obliczeń wykonywanych lokalnie
4. pełną synchronizację wszystkich wierzchołków – w każdym kroku algorytmu wszystkie wierzchołki jednocześnie wykonują obliczenia lokalne w czasie zerowym, a następnie wymieniają się komunikatami w czasie jednostkowym – taki krok wykonany równolegle przez wszystkie wierzchołki nazywamy *rundą*,

5. brak globalnej wiedzy o systemie, znane są tylko: lokalny stan wierzchołka, sąsiedzi w grafie systemu,
6. struktura systemu nie zmienia się w trakcie działania algorytmu,
7. każdy wierzchołek ma swój unikalny identyfikator,
8. rozmiar przesyłanych komunikatów nie jest ograniczony,
9. liczba operacji obliczeniowych wykonywanych lokalnie nie jest ograniczona

Model obliczeń all-port jest modelem bardzo silnym. Większość przyjętych tu założeń nie wydaje się realistyczna, jednak jest to model użyteczny i często stosowany do analizy algorytmów rozproszonych. Nie ma pełnej zgody, co do wszystkich założeń modelu, co za chwilę omówimy dokładniej.

Niezawodność

W punkcie 1 przyjęliśmy, że kanały komunikacyjne są niezawodne, z własnością *fifo* i nieograniczoną pojemnością. Zapewnienie poprawnej komunikacji jest oddzielnym zagadnieniem i zakładamy, że jest realizowane w niższej warstwie dostępnego protokołu.

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 5 z 23
Powrót
Full Screen
Zamknij
Koniec

Jednostkowy czas przesłania komunikatu

Czas przesłania komunikatu w systemach rzeczywistych różni się w zależności od jego długości, rodzaju połączenia, czy aktualnych warunków panujących w systemie. Precyzyjne uwzględnienie tych wszystkich czynników wydaje się bardzo trudne, dlatego przyjmujemy czas jednostkowy.

Czas obliczeń lokalnych

Czas wykonywania prostych obliczeń jest niewspółmiernie mały w stosunku do czasu potrzebnego na komunikację. Dlatego czas ten nie jest brany pod uwagę w analizie i porównaniach. Więcej wątpliwości budzi założenie przyjęte w punkcie 9. Można założyć wielomianowy czas wykonania w zależności od rozmiaru danych. Przyjęcie takiego ograniczenia eliminuje pewne patologie.

Bez tego ograniczenia można na przykład uzasadnić, że każdy problem algorytmiczny jest rozwiązywany w czasie proporcjonalnym do średnicy grafu systemu:
Najpierw dane o całym grafie zbieramy w jednym wierzchołku, następnie bez upływu czasu znajdujemy tam rozwiązanie optymalne po czym rozsyłamy odpowiedź do wszystkich węzłów sieci. Z drugiej strony, jeśli wielomian jest wysokiego stopnia to czas wielomianowy może i tak być zbyt długi do zastosowań. Ponadto typowo autorzy prac zwykle podkreślają prostotę i niską złożoność obliczeń lokalnych.

Synchronizacja i pomiar czasu

Przez pomiar czasu w modelu all-port rozumiemy wyłącznie liczbę rund jaka jest potrzebna do wykonania algorytmu.
Założenie o pełnej synchronizacji systemu jest wygodne przy analizie złożoności obliczeniowej algorytmu. Natomiast często nie jest konieczne dla jego poprawności. Wielu autorów podkreśla, że ich algorytmy działają w systemie asynchronicznym. Powstał też pomysł zdefiniowania tak zwanej *rundy asynchronicznej*.

Model one-port

Dodatkowym rygorem może być uniemożliwienie w ciągu jednej rundy wymiany komunikatów z więcej niż jednym sąsiadem. W tym wypadku mówimy o modelu *one-port*.
Można ograniczyć model tak, by w ciągu jednej rundy każdy wierzchołek mógł wysłać do wszystkich sąsiadów tylko jedną, tę samą wiadomość (ang. broadcast).

Zmienne globalne

Wiedza lokalna jest jedną z podstawowych różnic między systemami rozproszonymi i równoległymi. Przyjęcie w każdym z wierzchołków wiedzy o jakimkolwiek parametrze systemu, takim jak liczba wierzchołków lub maksymalny stopień, osłabia uzyskany wynik. Nie znaczy to jednak, że założenia takie są z gruntu nierealistyczne. Wiedza o niektórych parametrach może być znana a priori, co może wynikać na przykład ze specyfiki sprzętu użytego do budowy systemu, a w niektórych wypadkach można przyjąć wartości szacunkowe.

Struktura statyczna

W modelu all-port przyjmujemy, że system się nie zmienia w trakcie działania algorytmu. Ani nie przybywa, ani nie ubywa zarówno procesów jak i kanałów komunikacyjnych. Model, w którym zakłada się strukturę dynamiczną, jest oczywicie mniej restrykcyjny, a więc wynik uzyskany w takim modelu jest mocniejszy.

Identyfikatory

Ze względu na możliwość występowania symetrii konieczne jest zastosowanie mechanizmów do ich łamania. Założenie przyjęte w modelu (punkt 8) o istnieniu unikalnych identyfikatorów jest założeniem silnym. Słabsza wersja zakłada unikalność tylko w sąsiedztwie każdego wierzchołka.

2. Wybór lidera w cyklu c. d.

Znamy prosty algorytm dla cyklu, w którym wysyłanych jest $O(n^2)$ wiadomości. Pokażemy teraz metodę, która gwarantuje $O(n \log n)$ komunikatów.
Przez k-sąsiedztwo wierzchołka p_i rozumiemy zbiór wierzchołków w odległości nie większej niż k od p_i. Dla cyklu k-sąsiedztwo zawiera dokładnie $\min n, 2k + 1$ wierzchołków.

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 14 z 23
Powrót
Full Screen
Zamknij
Koniec

Algorytm przebiega etapami.
W l-tym etapie każdy z wierzchołków próbuje zostać liderem w swoim 2^l-sąsiedztwie. Tylko wierzchołki którym się to uda, kontynuują działanie w kolejnym etapie.
Każda wiadomość zawiera trzy pola:

- id - identyfikator nadawcy,
- l - numer etapu,
- hc - licznik przesłań.
- f - znacznik powrotu

Na początku każdej fazy każdy aktywny wierzchołek rozsyła w obu kierunkach wiadomości z własnym identyfikatorem, numerem etapu, licznikiem przesłań ustawionym na 0 i nieustawiną flagą powrotu. Tak jak w poprzednim algorytmie przekazywana dalej jest tylko otrzymana wiadomość o identyfikatorze większym niż własny. Jeśli $l = 2^{hc}$ ustawiany jest f, a wiadomość odsyłana z powrotem.

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 16 z 23
Powrót
Full Screen
Zamknij
Koniec

Lemat 1 *Dla $l > 1$ liczba aktywnych procesorów jest mniejsza lub równa od $n/2^{l-1}$.*

Ponadto każda z dwóch wiadomości wysłanych przez tymczasowego lidera przechodzi odległość $2 * 2^l$, co w sumie daje $2 * 2 * 2^l * n/2^{l-1} = 8n$ wiadomości w każdej fazie.

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 17 z 23
Powrót
Full Screen
Zamknij
Koniec

Wybór lidera w grafie pełnym

Idea algorytmu

Każdy wierzchołek „budzi się”, po czym próbuje zwerbować jak najwięcej wierzchołków do swojego „królestwa”. Mniejsze królestwo rozpada się, napotykając większe.

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 18 z 23
Powrót
Full Screen
Zamknij
Koniec

Podamy pseudokod dla wierzchołka p_i używając następujących komunikatów:

- **Collect(j,s)**: wiadomość do p_j - mam już s wierzchołków w swoim królestwie, więc i ty przyłącz się do mnie.
- **Join(j,s)**: o wielki p_j przyłączam się do twego królestwa.
- **Check(j,s)**: sprawdź, czy twoje królestwo jest większe niż królestwo p_j z s wierzchołkami.
- **Ack(j,s)** Potwierdzenie, że królestwo p_j jest większe od naszego.

Oraz zmiennych lokalnych:

- K - królestwo p_i w formie identyfikatora posiadacza, inicjalnie pusty ($\perp$).
- SK - rozmiar K, jeśli $K = i$
- *waiting* - wierzchołek, który chce przeciągnąć p_i do swego królestwa.

Algorytm 1: Wybór lidera w grafie pełnym

if procesor sam się obudził **then**
  $K = i$
  $KS = 1$
  $waiting = i$
  wyślij $Collect(i, 1)$ do któregoś z sąsiadów
end if

if odebrano $Collect(j, s)$ **then**
  if $K = \perp$ **then**
    $K = j$
    wyślij $Join(j, s)$ do p_j
  else
    $waiting = j$
    wyślij $Check(j, s)$ do K
  end if
end if
if odebrano $Check(j, s)$ od p_l **then**
  if $K < s$ **then**
    $waiting = \perp$
    $K = \perp$
    wyślij $Ack(j, s)$ do p_l
  end if
end if

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 22 z 23
Powrót
Full Screen
Zamknij
Koniec

if odebrano $Ack(j, s)$ od p_l **then**
  $waiting = \perp$
  wyślij $Join(j, s)$ do p_j
end if
if odebrano $Join(i, s)$ od p_l **then**
  if $waiting \neq \perp$ **then**
    $KS = KS + 1$
    wyślij $Collect(j, KS)$ do p_{l+1}
  end if
end if