Zusatznotizen Vorlesung Verteilte Algorithmen vom 29. 11. 2001
Prof. Dr. R. Wattenhofer

Untere Schranke beim Gegenseitigen Ausschluss

Gegeben sind n Prozesse P_1 bis P_n. Wir betrachten „statische“ Algorithmen: d.h. ein Prozess P_i muss immer die unveränderliche Menge von Prozessen R_i um Erlaubnis fragen, um in den kritischen Abschnitt einzutreten. Wir können die Korrektheit nur garantieren, wenn für alle i,j gilt $R_i \cap R_j \neq \emptyset$.

Was ist ein effizienter Algorithmus? max_i |R_i| möglichst klein?! (Zentraler Manager P_1 (R_i = {1}) ist gueltig, aber sicher nicht effizient...)

Oft wird deshalb verlangt, dass der Algorithmus „symmetrisch“ sein soll. Was heisst symmetrisch? Alternativ können wir sagen: es soll keinen Prozess geben, der in zu vielen Mengen R_i vorkommt. Formal heisst das: Wir definieren c_i [j] = 1, falls $j \in R_i$, sonst c_i [j] = 0 und C [j] = sum_i c_i [j]. Der am häufigsten vorkommende Prozess tritt dann in C Mengen auf mit C = max_i C [j]. Wir moechten C minimieren. Aber...

Theorem: $C > \sqrt{n}$

Beweis: Per Definition ist sum_i c_i [j] = C [j] ≤ C. Das dies für jedes j gilt ist sum_i |R_i| = sum_i sum_j c_i [j] = sum_j sum_i c_i [j] ≤ Cn. Dann gibt es ein i* mit |R_i*| ≤ C. Da C [j] ≤ C kann jeder Prozess $j \in R_i^*$ maximal in (C[j]-1) anderen Prozessen R_j auftreten. Das heisst die Anzahl der Prozesse n ist beschränkt durch:

n ≤ 1+(C-1) |R_i*| ≤ 1+(C-1)C = C^2-C+1; mit C>1 wird das zu n ≤ C^2-C+1<C^2,
und das Theorem folgt.

Verteiltes Zählen

Ist ein gegenseitiger Ausschluss immer notwendig? Ein gegenseitiger Ausschluss zerstört die Parallelität (und damit die Existenzberechtigung?) des verteilten Algorithmus. Deshalb wollen wir ihn wenn immer möglich vermeiden. Beispiel: Verteiltes Zählen. (Anwendungen: Dynamische Lastverteilung, Stack, Queue,)

Jeder Prozess hat eine lokale Variablen v_i;
Dann gibt es eine „globale“ Variable V, die zu Beginn mit 0 initialisiert ist.

```
Prozess i:
Loop
  Beginn kritischer Abschnitt
    v_i ← V;
    V ← V + 1;
  Ende kritischer Abschnitt
  Beginn lokale Berechnung (Beispiel: SETI@home)
    Lade Datenblock v_i von Server v_i MOD n;
    berechne die Fouriertransformation dieses Datenblocks;
    Wenn Aliens gefunden: schicke email an National Enquirer;
  Ende lokale Berechnung
Zurück zum Loop
```

Problem: Wenn die Anzahl der Prozesse wächst, dann wird der kritische Abschnitt zum Nadelöhr; irgendwann nützt es nichts, wenn man mehr Prozesse dazufügt. Doch vielleicht geht es ja ohne gegenseitigen Ausschluss?

Abstrakter Datentyp „Verteilter Zähler“:

- Variable V, initialisiert mit 0.
- Eine Zugriffsfunktion INC(V) liefert Wert von aktuellem V und inkrementiert V um 1.

Korrektheit (wenn System ruhig, d.h. kein Aufruf von INC gestartet aber noch nicht beendet)

- Nach k erledigten Aufrufen von INC wurden genau die Werte 0, 1, ..., k-1 verteilt, also keiner doppelt und keiner gar nicht.
- Sonst noch was?

Lösungen? Zentraler Zähler, jeder Prozess kennt den Zahlerwert V, c Zähler??

Zählnetzwerk (Bitonic Counting Network)

(siehe Folien)

Ein Balancer (Flip-Flop) und ein „kleine Zähler“ am Ende jedes Drahtes kann durch einen Prozessor implementiert werden, die Drähte zwischen den Prozessoren symbolisieren dann Nachrichtenkanäle. Ein Zählnetzwerk ist dezentral. Es ist fehlertoleranter, und unter hoher Belastung effizienter als ein zentraler Zähler (die Tiefe eines Bitonischen Zählnetzwerkes mit n Drähten ist log^2 n; es gibt aber auch komplizierte Zählnetzwerke mit log n Tiefe).

(An der Tafel: Rekursive Definition und Beweisidee, warum das Zählnetzwerk nach obiger Definition korrekt zählt.)

Man könnte eine verschärfte Korrektheit verlangen (muss immer gelten, nicht nur wenn das System ruhig ist):

- Linearisierbarkeit: Ist ein Aufruf A von INC beendet, bevor wir einen Aufruf B von INC starten, dann muss V_A < V_B sein. Problem: Nicht-triviale Zählnetzwerke sind nicht linearisierbar.

Meine liebste offene Frage der verteilten Algorithmik: Was kann „verteilt“ (im Sinne von dezentral) berechnet werden, und was beweisbar nicht?