Organisatorisches

- Webseite zur Vorlesung:

  `http://i11www.ira.uka.de/teaching/SS_05/algosens/`

  - Folien zur Vorlesung und Übung (soweit verfügbar)
  - Übungsblätter
  - Referenzen auf weiterführende Literatur
  - Ankündigungen zur Vorlesung

Inhalt

- Literatur
  - H. Attiya, J. Welch: *Distributed Computing*. Wiley, 2004.
  - R. G. Gallager, P. A. Humblet, and P. M. Spira: A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems* 5:1, 66-77, 1983.

Verteilte Systeme

Definition 1
Ein verteiltes System ist eine Menge von selbständigen *Recheneinheiten*, die miteinander *kommunizieren* können.

Verteilte Systeme

- Beispiele
  - Multiprozessorrechner
  - Cluster von Workstations
  - Internet, P2P Netze, Seti@home
  - Sensor- und Ad Hoc-Netze
- Grundlegende Schwierigkeiten
  - Asynchrone Kommunikation
  - Begrenztes lokales Wissen
  - Ausfälle

Theorie des Verteilten Rechnens

- Ziele sind ähnlich wie bei sequentiellem Rechnen, z.B. Algorithmik, Komplexitätstheorie, usw.
- Problem: Kein einheitliches Modell für verteilte Systeme
  - Unterschiedliche Kommunikationsparadigmen
  - Sind Recheneinheiten unterscheidbar?
  - ...
- Komplexitätsmaße
- Basisalgorithmen
- Schwierigkeit von Problemen: oft Negativresultate

Kommunikationsparadigmen

- Synchron ↔ Asynchron
- Nachrichtenbasiert ↔ Shared Memory
  - Synchrone Kommunikation über Shared Memory entspricht dem PRAM Modell
  - Wir betrachten im Folgenden nur Modelle für nachrichtenbasierte („message passing") verteilte Systeme

Beispiel: Broadcast

- Gegeben
  - Kommunikationsgraph $G = (V, E)$
  - Aufspannender Baum mit Wurzel $r \in V$
    - Jeder Knoten v kennt den Nachbarn, der im Baum zur Wurzel zeigt (`parent`(v))
    - Jeder Knoten v kennt die Nachbarn, die im Baum von der Wurzel weg zeigen (`children`(v))
  - Eine Nachricht M an einem Knoten r
- Problem
  - Die Nachricht M soll an alle Knoten des Netzes übermittelt werden

Formales Modell eines verteilten Systems (I)

- n Prozessoren („Recheneinheiten") $p_1, \ldots, p_n$
- Zwei Prozessoren können durch einen bidirektionalen Kommunikationskanal verbunden sein
- Kommunikationsgraph („Topologie", Netz)
  - Knoten: Prozessoren $p_1, \ldots, p_n$
  - Kanten: Kommunikationskanäle $e_{ij} = \{p_i, p_j\}$ $(i \neq j)$
  - Der Grad (Anzahl inzidenter Kanten) von p_i sei d_i
  - Der Graph ist ungerichtet

Formales Modell eines verteilten Systems (II)

- Jeder Prozessor p_i hat Zustandsmenge Q_i, darunter ausgezeichnete Anfangs- und Endzustände
- Kommunikationspuffer $\texttt{outbuf}_i[l]$ und $\texttt{inbuf}_i[l]$ $(1 \leq l \leq d_i)$, jeder dieser Puffer besteht aus einer Folge von Nachrichten
- Am Anfang sind die $\texttt{inbuf}_i[l]$ leer
- Kommunikationspuffer können auch in die Zustände Q_i modelliert werden

Beispiel: Broadcast (Zustände)

- Die Zustände eines Prozessors p_i beinhalten
  - $\texttt{parent}(p_i)$=$\texttt{parent}_i$, ein Prozessorindex oder `nil`
  - $\texttt{children}(p_i)$=$\texttt{children}_i$, eine Menge von Prozessorindizes
  - $\texttt{terminated}_i$, eine `boolean` Variable
- Anfangszustände: $\texttt{parent}_i$ und $\texttt{children}_i$ gemäß aufspannendem Baum initialisiert, $\texttt{terminated}_i = \texttt{false}$
- $\texttt{outbuf}_r[j]$ enthält M für alle $j \in \texttt{children}_r$
- Alle anderen $\texttt{outbuf}_i$ sind leer

Verteilter Algorithmus

- Berechnungsschritt $\text{comp}(i)$ bei p_i
  - p_i befindet sich in Zustand $q_i \in Q_i$
  - Abhängig von q_i und $\texttt{inbuf}_i$ Zustandsübergang
  - Nach Berechnungsschritt ist $\texttt{inbuf}_i$ leer
  - Für jede inzidente Kante kann eine Nachricht erstellt und zu $\texttt{outbuf}_i[l]$ hinzugefügt werden
- Kommunikationsschritt $\text{del}(i, j, M)$: Sende M über e_{ij}
  - Eine Nachricht M wird von $\texttt{outbuf}_i[l_i]$ nach $\texttt{inbuf}_j[l_j]$ übertragen

Verteilter Algorithmus

- Konfiguration $C = (c_1, \ldots, c_n)$
  - c_i enthält Zustand q_i des Prozessors p_i sowie $\texttt{inbuf}_i$ und $\texttt{outbuf}_i$
  - Beschreibt vollständig den Zustand des ganzen Systems zu einem bestimmten Zeitpunkt
- Ausführung eines verteilten Algorithmus
  - Unendliche Folge von Konfigurationen und Schritten: $C_1, \phi_1, C_2, \phi_2, \ldots$
  - Eine gültige Ausführung muss gewisse Bedingungen erfüllen

Beispiel: Broadcast (Algorithmus)

- Berechnungsschritt bei p_i:
  - Falls $M \in \texttt{inbuf}_i[k]$ für ein k
    - füge M zu $\texttt{outbuf}_i[j]$ hinzu, für alle $j \in \texttt{children}_i$
    - setze $\texttt{terminated}_i = \texttt{true}$
  - Falls $i = r$ und $\texttt{terminated}_r = \texttt{false}$, so setze $\texttt{terminated}_r = \texttt{true}$
  - Ansonsten wird nichts getan

Asynchrone Kommunikation

- Keine Garantien bzgl. der Dauer einer Nachrichtenübermittlung
- Algorithmus ohne zeitliche Koordinierung („asynchron")
- Sei eine Ausführung $C_1, \phi_1, C_2, \phi_2, \ldots$
- Gültigkeitsbedingungen:
  - C_{k+1} muss durch Schritt ϕ_k aus C_k hervorgehen
  - Jeder Prozessor hat unendlich viele Berechnungsschritte
  - Jede Nachricht wird übermittelt (Achtung: „eventually")
- C_1 besteht aus Initialzuständen mit leeren $\texttt{inbuf}_i$

Synchrone Kommunikation

- Rundenbasiert
  1. Jeder Prozessor kann eine Nachricht zu jedem Nachbarn senden
  2. Diese Nachrichten werden sofort zugestellt
  3. Jeder Prozessor führt einen Berechnungsschritt aus
- Spezialfall einer asynchronen Kommunikation:
  Eine Runde besteht aus
  1. Kommunikationsschritten bis alle $\texttt{outbuf}_i$ leer sind
  2. Einem Berechnungsschritt pro Prozessor
- Die Ausführung ist durch die Anfangskonfiguration C_1 festgelegt

Korrektheit eines verteilten Algorithmus

- Jede gültige Ausführung ist unendlich
- Jeder Prozessor hat eine Teilmenge von Endzuständen
- Aus Endzuständen können nur Endzustände erreicht werden
- Eine gültige Ausführung ist beendet, wenn alle Prozessoren in einem Endzustand sind
- Ein verteilter Algorithmus ist korrekt, wenn alle gültigen Ausführungen nach endlich vielen Schritten enden und danach stets eine korrekte Lösung des Problems vorliegt

Beispiel: Broadcast (Korrektheit)

Lemma 2
Der Broadcast Algorithmus ist korrekt, d.h. jede Ausführung ist nach endlich vielen Schritten beendet und jeder Prozessor hat die Nachricht M erhalten.

Beispiel: Broadcast (Korrektheit)

Beweis.

- Ein Prozessor ist im Endzustand g.d.w. er M erhalten hat
- Annahme: Es gibt einen Prozessor, der nicht terminiert
  - Sei p_i solch ein Prozessor mit minimaler Höhe im Baum
  - $p_i \neq r$, denn im ersten Berechnungsschritt von r terminiert r
  - Der Vorgänger von p_i im Baum ist terminiert
  - Dann hat der Vorgänger aber M an p_i geschickt
  - Da alle Nachrichten übermittelt werden und jeder Prozessor unendlich viele Berechnungsschritte hat, wird auch p_i nach endlich vielen Schritten M erhalten und terminieren
  - Widerspruch zur Annahme, also terminiert jeder Prozessor und erhält somit auch M

□

Nachrichten-Komplexität

- Maximale Anzahl an gesendeten Nachrichten über alle gültigen Ausführungen des Algorithmus
- Kann bei asynchroner und synchroner Kommunikation angewandt werden

Beispiel: Broadcast (Nachrichten-Komplexität)

Lemma 3
Der Broadcast Algorithmus hat Nachrichtenkomplexität $n - 1$.

Beweis.

- Die Nachricht M wird genau einmal pro Kante im aufspannenden Baum versandt
- Jeder Baum mit n Knoten hat $n - 1$ Kanten

□

Zeit-Komplexität

- Synchrone Kommunikation
  - Laufzeit: Maximale Anzahl der Runden
- Asynchrone Kommunikation
  - Maximale Übermittlungsdauer ist 1
  - Zeitliche Ausführung
    - Jedem Schritt wird ein nichtnegativer Zeitpunkt zugeordnet
    - Zeit startet bei 0, darf nicht abnehmen und nicht beschränkt sein
    - Pro Prozessor ist die Zeit strikt ansteigend
  - Dauer einer Nachrichtenübermittlung: Zeit zwischen Einfügen in $\texttt{outbuf}_i[l_i]$ und Lesen aus $\texttt{inbuf}_j[l_j]$
  - Normierung der maximalen Dauer auf 1
  - Laufzeit: Maximale Zeit bis zur Beendigung des Algorithmus

Beispiel: Broadcast (synchrone Zeitkomplexität)

Lemma 4

Bei jeder gültigen Ausführung im synchronen Modell empfängt jeder Prozessor mit Entfernung t von r im aufspannenden Baum die Nachricht M in Runde t.

Beispiel: Broadcast (synchrone Zeitkomplexität)

Beweis.

Induktion nach t.

- Induktionsanfang: $t = 1$. Jedes Kind von r empfängt M in der ersten Runde.
- Induktionsannahme: Jeder Prozessor mit Entfernung $t-1$ hat M in Runde $t-1$ empfangen.
- Induktionsschluss:
  - Sei p_i ein Prozessor mit Entfernung t im Baum
  - Der Vorgänger $\texttt{parent}_i = p_j$ von p_i hat also Entfernung $t-1$, und nach Induktionsannahme hat p_i die Nachricht M in Runde $t-1$ empfangen
  - Nach dem Algorithmus sendet p_j die Nachricht M in der darauffolgenden Runde an p_i.

□

Beispiel: Broadcast (synchrone Zeitkomplexität)

Lemma 5

Der Broadcast Algorithmus hat im synchronen Modell eine Zeit-Komplexität von h, wobei h die Höhe des aufspannenden Baumes ist.

Beweis.

- Die maximale Entfernung von der Wurzel im Baum ist die Höhe h des aufspannenden Baumes
- Der Beweis folgt somit aus Lemma 4

□

Problemstellung

- Voraussetzungen
  - Der Kommunikationsgraph ist ein Ring aus n Prozessoren
  - Jeder Prozessor kennt seinen *linken* und *rechten* Nachbarn
- Problem
  - Genau ein Prozessor soll als *Leader* ausgezeichnet werden

Anonymität, Uniformität

Definition 6
Ein verteilter Algorithmus heißt uniform, wenn die Anzahl n der Prozessoren nicht verwendet wird.

Definition 7
Ein verteilter Algorithmus heißt anonym, wenn die Prozessoren keine unterschiedlichen IDs besitzen. (D.h., die Prozessoren führen identische Programme aus.)

Satz 8
Es gibt keinen anonymen Algorithmus für das Leader Election Problem in Ringen.

Beweis.

- Annahme: Es gibt einen solchen Algorithmus
- $O.B.d.A.$ sei der Algorithmus synchron
- Die Ausführung R des Algorithmus ist durch die Anfangskonfiguration festgelegt
- Da der Algorithmus anonym ist, sind die Zustandsübergangsfunktionen für alle Prozessoren identisch
- In jeder Runde k der Ausführung R sind alle Prozessoren am Ende der Runde im selben Zustand
  - Induktion über k
- Sobald ein Prozessor am Ende einer Runde ausgewählt wird, sind alle Prozessoren ausgewählt, also kann der Algorithmus nicht korrekt sein

□

„Symmetry Breaking"

- Jeder Prozessor p_i hat eindeutige $\texttt{ID}_i$
- Einfacher uniformer Algorithmus:
  - Am Anfang: jeder Prozessor p_i sendet $\texttt{ID}_i$ zum linken Nachbarn
  - Prozessor p_i empfängt eine ID vom rechten Nachbarn
    - Falls $\texttt{ID} > \texttt{ID}_i$: sende ID zum linken Nachbarn
    - Falls $\texttt{ID} = \texttt{ID}_i$: sende Nachricht „terminiere" zum linken Nachbarn und terminiere selbst als „Leader"
    - Ansonsten sende ID nicht weiter
  - Prozessor p_i empfängt Nachricht „terminiere"
    - terminiere als „Non-Leader"
- Nachrichten-Komplexität: $\mathcal{O}(n^2)$
- Bessere Algorithmen: $\Theta(n \log n)$
- Andere Möglichkeit: Randomisierung

- Voraussetzungen
  - Ungerichteter, zusammenhängender Kommunikationsgraph $G = (V, E)$
  - Kantengewichte $W : E \to \mathbb{R}$, paarweise verschieden
  - Asynchrone Kommunikation
  - Jeder Knoten kennt Kantengewichte inzidenter Kanten
- Problem
  - Bestimme einen aufspannenden Baum minimalen Gewichtes (Minimum Spanning Tree, MST)
- Motivation in Sensor- und Ad Hoc-Netzen
  - Kantengewichte entsprechen Energieaufwand für Kommunikation
  - Verwende MST als aufspannenden Baum bei Broadcast
  - Minimaler Energieaufwand

Definition 9
Sei T ein Teilbaum („Fragment") eines Graphen $G = (V, E)$, der durch eine Teilmenge der Kanten $E_T \subset E$ induziert wird. Dann heißt eine Kante e ausgehend aus T wenn genau einer der beiden Endknoten von e zum Teilbaum T gehört.

Lemma 10
Sei ein Teilbaum T eines MST induziert durch eine Kantenmenge E_T und e eine aus T ausgehende Kante minimalen Gewichtes. Dann ist der durch $E_T \cup e$ induzierte Baum ebenfalls ein Teilbaum eines MST.

Sequentieller Algorithmus von Prim (Wdh.)

1. Wähle einen beliebigen Startknoten und betrachte diesen als „grünen Baum"
2. Wiederhole den folgenden Färbungsschritt $|V| - 1$ mal
3. Wähle eine ungefärbte aus dem grünen Baum ausgehende Kante minimalen Gewichtes und färbe sie grün.

- Zu sequentiellen MST-Algorithmen siehe auch Kapitel 4 im Skript zur Vorlesung „Entwurf und Analyse von Algorithmen":
  `http://i11www.ira.uka.de/algo/teaching/scripts/sources/ad.pdf`