Grundlagen: Überblick

- Verteilte Systeme
  - Definition
  - Grundbegriffe
  - Kommunikation
- Klassifikation von Fehlern
  - Begriffe
  - Fehlerarten
- Analyse von Algorithmen
  - Korrektheit
  - Komplexität

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Verteilte Systeme: Definition

- Definition von Leslie Lamport

  "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

  ⇒ *"If you do not need a distributed system, do not distribute"* (Veríssimo)

  ⇒ Fehlertoleranz von verteilten Systemen ist eine sehr wichtige Eigenschaft, die auch heute noch in vielen Systemen fehlt!

- Definition von Andrew Tanenbaum

  "Ein verteiltes System ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen."

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Verteilte Systeme: Merkmale

- Mehrere, unabhängige Rechner
  - können unabhängig voneinander ausfallen
- Verbunden durch ein Netzwerk
  - Interaktion nur durch Nachrichtenaustausch möglich

    ⇒ *Unterschied zu Parallelrechnern!*
  - Netzwerk unzuverlässig, mit variablen Nachrichten-verzögerungen, moderate Übertragungsgeschwindigkeit im Vergleich zu Multiprozessor-/Multicomputersystemen
- Kooperation der Knoten
  - Beteiligte Knoten interagieren, um gemeinsam eine Aufgabe zu lösen oder einen Dienst anzubieten.

    ⇒ *Unterschied zu einem Rechnernetz*

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Verteilte Systeme: Definition (V. Garg)

- Keine gemeinsame Uhr
  - Uhrensynchronisation schwierig aufgrund nicht bekannter Verzögerungszeiten bei der Kommunikation, physikalische Uhren daher nur selten zur Synchronisation verwendet
- Kein gemeinsamer Speicher
  - Kein einzelner Knoten kennt den vollständigen globalen Zustand. Es ist daher schwierig, globale Eigenschaften des Systems zu beobachten.
- Keine akkurate Ausfallerkennung
  - Es ist in einem asynchronem verteiltem System (d.h. keine obere Schranke für Komm'zeiten bekannt) unmöglich, langsame von ausgefallenen Prozessen zu unterscheiden

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Verteilte Systeme: Bezeichnungen

- *Knoten*, *Rechner, Prozessor, Prozess*:

  Meist synonym gebraucht; bezeichnen eine einzelne aktive Instanz im verteilten System, die mit den anderen Instanzen durch Nachrichtenaustausch interagiert

- *Globaler Zustand:*

  Verteilter Zustand des Systems zu einem Zeitpunkt der realen Zeit, ausgedrückt durch einen Vektor $S = [S_1, ..., S_n]$ aus den Zuständen S_i aller n im System vorhandenen Knoten

- *Schritt:*

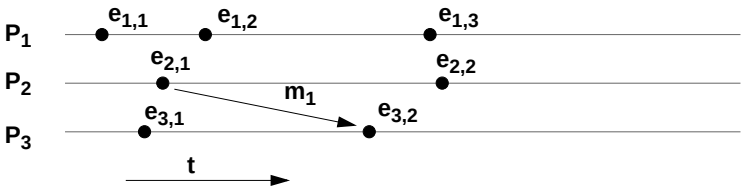
  Atomarer Übergang von einem globalen Zustand in einen Folgezustand

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Verteilte Systeme

- Zeit-Raum-Diagramm

  Graphische Darstellung von (lokalen Ereignisse und) Interaktionen aller Prozessoren



  Konvention für Bezeichner:

  - Prozesse: Grossbuchstaben (A, B, C, ... oder P_1, P_2, P_3, ...)
  - Ereignisse: Kleinbuchstaben (a, b, c, ... oder e_1, e_2, e_3, ...)
  - Nachrichten: Kleinbuchstaben (m_1, m_2, ...)

    Übertragungsrichtungen als (Sender, Empfänger) (z.B. $m_1^{2,3}$)

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Eigenschaften des Kommunikationssystems

- Punkt-zu-Punkt-Verbindung
  - ungeordnet
  - reihenfolgeerhaltend: FIFO-Ordnung (First In First Out)
- Gruppenkommunikation/Multicast
  - ungeordnet
  - FIFO-Ordnung
    - Nachrichten, die von einem Prozess in einer bestimmten Reihenfolge erzeugt wurden, werden von anderen Knoten in genau dieser Reihenfolge empfangen/bearbeitet
    - Nachrichten, die von unterschiedlichen Prozessen gesendet werden, können in unterschiedlicher Reihenfolge empfangen werden

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Eigenschaften des Kommunikationssystems

- Gruppenkommunikation/Multicast: ...Fortsetzung
  - (Potentielle) kausale Ordnung (Präzedenzordnung)
    - Nachricht a trifft bei allen vor b ein, falls b von a beeinflusst werden konnte (Notation: a $\rightarrow$ b)
    - keine totale Ordnung
  - Konsistente totale Ordnung
    - Es wird auf allen Ereignissen eine totale Ordnung definiert, die auch allen Prozessen des Systems bekannt ist
  - Ordnung nach Realzeit (Physikalische Ordnung)
    - kaum realisierbar, keine totale Ordnung

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

“Synchron” und “Asynchron”

Oft sind hier unterschiedliche Dinge damit gemeint:

- Entfernte Methodenaufrufe (send/receive bzw. request/reply-Interaktion):
  - *synchron*: Aufrufer wartet (blockierend) auf Ergebnis
  - *asynchron*: Anrufer wartet nicht (nicht-blockierend)
- Ausführung von verteilten Aktivitäten:
  - *synchron*: Synonym zu “gleichzeitig”: Aktivitäten werden mit Synchronisation untereinander ausgeführt (durch gemeinsame Uhren oder andere Synchronistationsmechanismen gesteuert)
  - *asynchron*: Keine Synchronisation vorhanden
- Koordinierung:
  - “Synchronisierung” als Synonym gebraucht; “nicht gleichzeitig”

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

“Synchron” und “Asynchron”

Im Kontext von verteilten Algorithmen übliche Definition

- *synchron*: Es sind feste Aussagen zur maximalen Dauer von Aktionen bzw. zur Laufzeit von Nachrichten möglich.
  - Algorithmen können dadurch verteilt in synchronisierten Runden ablaufen
  - Vorteilhafte Eigenschaft, weil damit Entscheidungen aufgrund des Ablaufs von Zeit getroffen werden können.
    Beispiel: Wird eine erwartete Nachricht nicht innerhalb der maximalen Laufzeit empfangen kann sicher auf einen Fehler (Knotenausfall, Nachrichtenverlust) geschlossen werden
- *asynchron*: Nachrichten können prinzipiell beliebig lang vom System verzögert werden.
  - Aus dem zeitlichen Ablauf (z.B. Timeouts) können keine sicheren Informationen gewonnen werden.

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

“Synchron” und “Asynchron”

- Für synchrone Systeme lassen sich oft einfache Algorithmen finden, allerdings werden diese oft nicht einsetzbar sein...
- Algorithmen für asynchrone Systeme lassen sich bei beliebigen Kommunikationsnetzen immer einsetzten. Sie existieren leider nicht immer, oder sind sehr komplex und ineffizient...
- Daher: Oft lohnt es sich, auch Zwischenpositionen (partielle Synchronität) zu betrachten:
  - Zeitliche Schranken existieren, aber deren Wert ist nicht bekannt
  - Bekannte zeitliche Schranken existieren, aber sie gelten nicht immer

Klassifikation von Fehlern

- "Fehler" wird in englischsprachiger Literatur oft differenziert betrachtet:
  - *Fault:* Unerwünschter Zustand, der zu einem "Error" führen kann; verursucht beispielsweise durch externe Störungen, durch Abnutzung bedingte Probleme oder durch Design-Fehler
  - *Error:* Systemzustand, der nicht den Spezifikationen entspricht, verursacht durch einen "Fault"
  - *Failure:* System erbringt nach außen nicht mehr den Dienst, den es erbringen sollte

    "An error is a manifestation of a fault in a system, which could lead to system failure" (Singhal/Shivaratri)

Klassifikation von Fehlern

- Ein "Failure" kann zugleich wieder auch "Fault" auf einer anderen Ebene sein!
- Beispiele:
  - Ein physischer Defekt im RAM (fault) kann dazu führen, dass Daten im Speicher verfälscht werden (error), und daher falsche Daten von einem Prozess gelesen werden (failure)
  - Die falschen Daten (fault) können dazu führen, dass der Prozess einen ungültigen Zustand erreicht (error), und dadurch "abstürzt" oder seinen Dienst nicht mehr korrekt erbringt (failure)
  - In einem verteilten System kann der Ausfall eines Knotens (fault) dazu führen, dass ein verteilter Algorithmus einen unerwarteten Zustand einnimmt (error), und er dadurch seine gewünschte Funktionalität nicht mehr erbringen kann (failure)

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Klassifizierung von Fehlern

- gutmütige Fehler (benign faults)
  - “Crash Failure”: Ein Knoten fällt plötzlich aus, d.h. bis zum Zeitpunkt t verhält er sich spezifikationsgerecht, danach werden keine Aktionen mehr durchgeführt.
    - Auch als “Halting Failure” oder “Fail Stop” bezeichnet. Oft bedeuten diese Begriffen aber zusätzlich, dass alle nicht ausgefallenen Knoten den Ausfall sicher feststellen können
  - Omission Failure: Manche Aktionen werden nicht ausgeführt
  - Timing Failure: Im Prinzip spezifikationsgerechtes Verhalten, aber manche Aktionen werden zu spät ausgeführt

  Allen gemeinsam: Der fehlerhafte Prozess führt keine Aktionen aus, die ein korrekter Prozess nicht machen würde

  Nicht immer realisitisch!

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Klassifizierung von Fehlern

- beliebige, "bösartige" Fehler (malicious faults)

  ⇒ Häufig als Byzantinische Fehler bezeichnet (nach Lamport, 1982)

  - Fehlerhafte Prozesse können beliebige Aktionen ausführen, und dabei auch untereinander kooperieren.
  - Häufig auch etwas eingeschränktes Modell
    - z.B. fehlerhafte Prozesse können beliebige mit polynomialer Komplexität berechenbare Aktionen ausführen (also u.a. keine digitalen Signaturen von korrekten Knoten fälschen!)
  - Modell, das alle beliebigen Arten von Fehler umfasst, z.B. auch gezielte Angriffe von aussen auf mein System

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg

Analyse von Algorithmen

- Korrektheit
  - Partielle Korrektheit *("Integrität", "safety")*: Wenn der Algorithmus terminiert, liefert er ein korrektes Ergebnis.
  - Totale Korrektheit: Partielle Korrektheit + der Algorithmus terminiert immer *("liveness")*
- Komplexität (nur ganz oberflächlich)
  - O-Notation: Die Komplexität O(f(n)) bedeutet, dass für die tatsächliche Komplexität p(n) folgendes gilt:

    Es gibt zwei Konstanten n_0, c, so dass für alle $n>n_0$ gilt:

    $p(n) \leq c*f(n)$
- Beachte: “eventually” vs. “eventuell”

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Lehrstuhl für Informatik 4, Universität Erlangen-Nürnberg