Verteilte Systeme

...für C++ Programmierer

Verteilte Synchronisation

by

Dr. Günter Kolousek

© Dr. Günter Kolousek

Creative Commons Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International Lizenz

Motivation

- 100 Programmierer entwickeln eine Anwendung
  - Übersetzen verteilt über viele Maschinen mit `make`
  - keine globale Zeit $\rightarrow$ kann sein, dass
    - einige Dateien nicht übersetzt werden,
    - die alte Version gelinkt wird

    und keine funktionsfähige Software vorliegt.
- Eine E-Mail, die beim Empfänger 5 Minuten vor dem Versendezeitpunkt ankommt!
  - Weiters: Empfänger antwortet und Antwort kommt noch immer 2 Minuten vor Absendezeit der ursprünglichen E-Mail an!

Möglichkeiten

- Beziehen der Atomzeit (absolute Zeit)
  - Sender DCF77 in Mainflingen
    - Genauigkeit zur Erkennung der Sekundenmarke ca. 100μs
    - Uhr hat ca. eine Abweichung von der „echten” Zeit von 1s in 30000 Jahren
  - per NTP
    - Genauigkeit liegt bei 10ms (über das Internet)
    - in Ö: bevtime1.metrologie.at, bevtime2.metrologie.at, time.metrologie.at
- Beziehen einer relativen Zeit
  - mehrere Computer müssen über eine gemeinsame Zeit verfügen

Verfahren von Cristian

”Absolute Zeit erforderlich”

- Ablauf
  1. Client sendet Nachricht an Server zum Zeitpunkt t_0
  2. Server empfängt Nachricht, verarbeitet diese und sendet Antwort an Client
  3. Client empfängt Nachricht zum Zeitpunkt t_1. Dies wird periodisch durchgeführt.
- Annahmen
  - Nachricht hin und zurück dauert ungefähr gleich lange
  - Server: zur Bearbeitung der Anfrage wird die Zeitdauer I benötigt
- Berechnung der Zeit: $t = t_s + (t_1 - t_0 - I)/2$

Verfahren von Cristian - 2

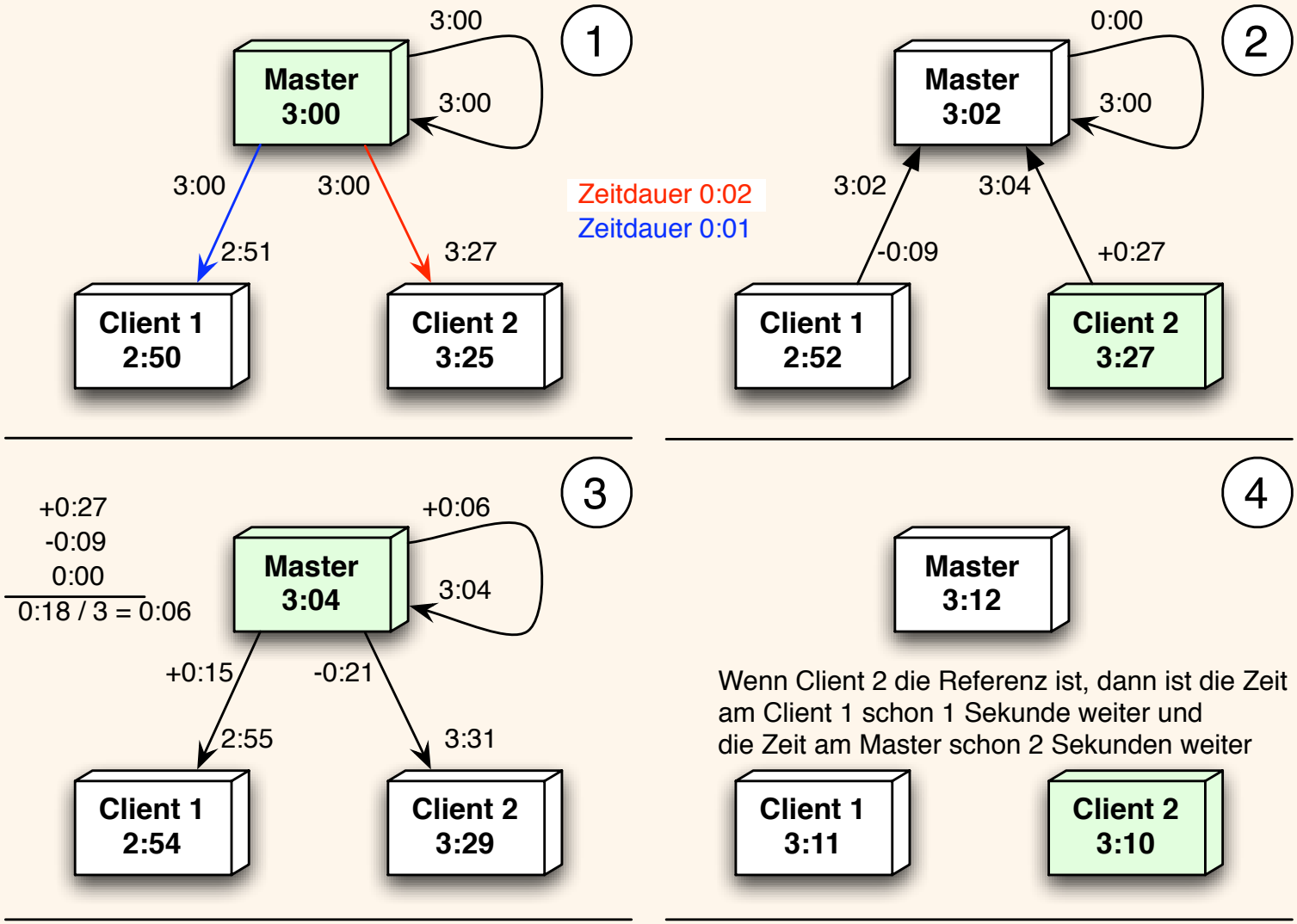
- Die Zeit könnte auch zurückgestellt werden! Das darf (normalerweise) nicht vorkommen
  - deshalb wird die Änderung schrittweise vorgenommen: Uhr wird langsamer gestellt bis Sollzeit erreicht ist.
- Die Zeit hängt von einem Server ab
  - könnte ausfallen (single point of failure)
  - könnte zu Performanceproblemen führen (Flaschenhals)
  - d.h. mehrere Server kontaktieren
    - Mittelwert bilden
    - fehlerhafte Server $\rightarrow$ Ausreißer eliminieren

Berkeley-Algorithmus

”Keine absolute Zeit notwendig”

- Ablauf
  1. Master fragt alle Slaves nach der aktuellen Zeit
  2. Master sammelt Antworten ein und berechnet Durchschnittszeit
     - Ausreißer werden eliminiert
  3. Master sendet Änderungsanforderung an jeden Slave
- Verwendet Verfahren von Cristian, um die Client-Zeiten zu berechnen

Berkeley-Algorithmus - 2



Globaler Status

- Status eines verteilten Systems zu einem Zeitpunkt
  - Sicht von oben!
- Setzt sich zusammen aus
  - lokalen Zustand aller Netzprozesse
  - Nachrichten, die gerade übertragen werden
- Kennt man den globalen Zustand $\rightarrow$ Deadlockerkennung möglich

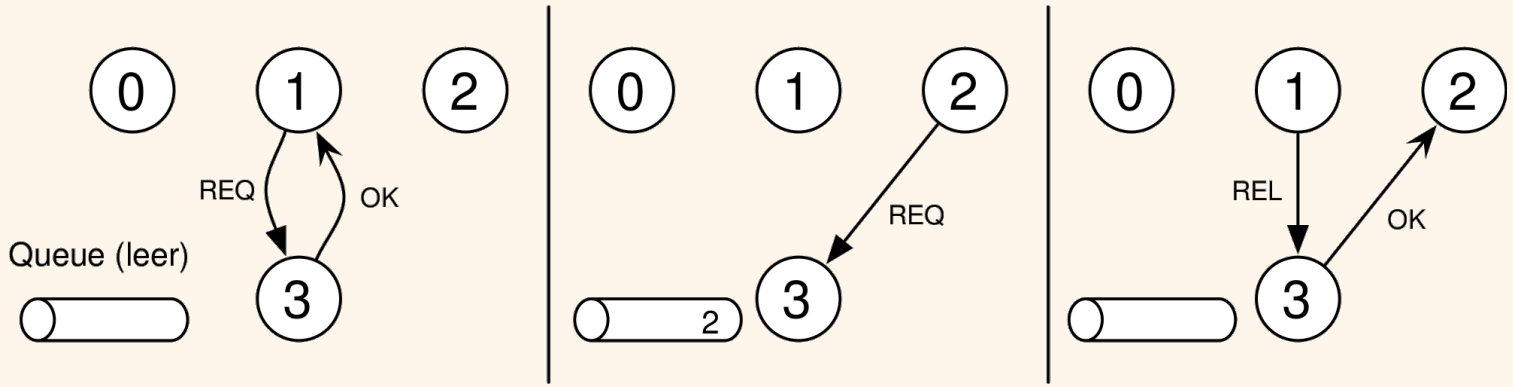
Wechselseitiger Ausschluss

- Zentralisierter Algorithmus
- Verteilter Algorithmus
- Tokenring Algorithmus

Zentralisierter Algorithmus

- ähnlich der Vorgangsweise im nicht verteilten System
  - zentraler Koordinator
- Ablauf
  1. Prozess will in kritischen Abschnitt eintreten und stellt Anforderung REQ an Koordinator, um auf Ressource zugreifen zu können.
  2. Noch kein anderer Prozess im kritischen Abschnitt?
     - ja: Koordinator sendet OK-Antwort zurück (Erteilung) und Prozess kann in kritischen Abschnitt eintreten.
     - nein: Koordinator stellt Anfrage in Queue und sendet *keine* Antwort
  3. Prozess verlässt kritischen Abschnitt und sendet Freigebenachricht REL an Koordinator
  4. Koordinator nimmt nächste Anforderung aus Queue und sendet OK-Antwort an entsprechenden Prozess

Zentralisierter Algorithmus - 2



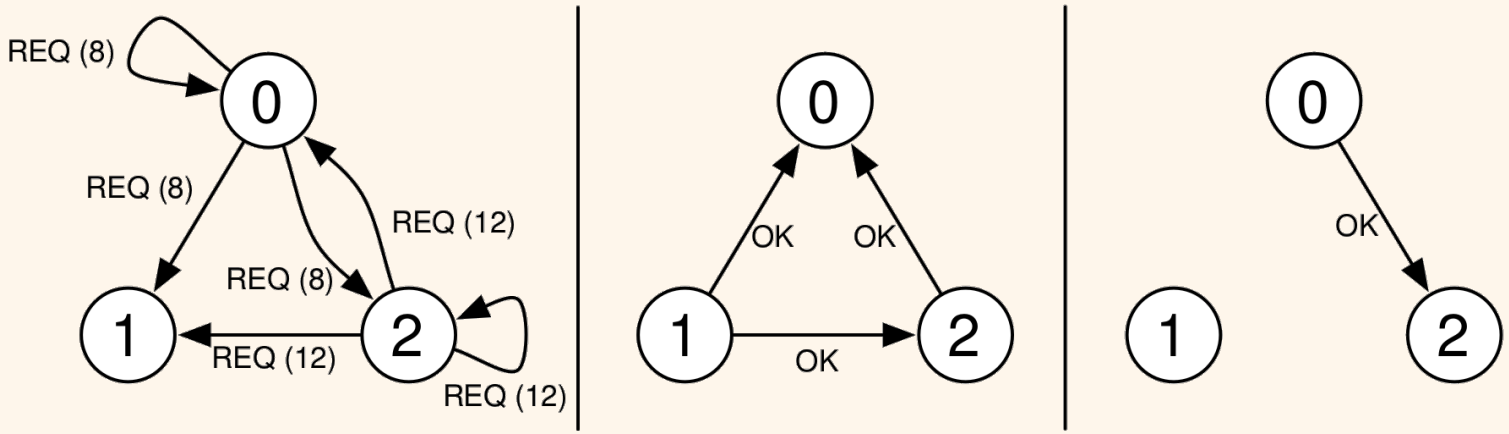
Zentralisierter Algorithmus - 3

- Vorteile
  - einfache Implementierung, nur 3 Nachrichten
  - faires Behandlung der Anfragen
- Nachteile
  - Koordinator ist zentraler Ausfallpunkt
  - ausgefallener Koordinator kann nicht erkannt werden, wenn dieser nach Annahme einer Anforderung ausfällt
  - Koordinator kann Leistungsengpass darstellen (Flaschenhals)
  - kein Koordinator (z.B. wenn ausgefallen), dann muss einer gewählt werden (Wahlalgorithmen)

Verteilter Algorithmus

- Versucht Nachteile des zentralisierten Algorithmus zu vermeiden
- Ablauf
  - Prozess erzeugt REQ Nachricht mit Prozessnummer und aktueller Zeit und sendet diese an alle anderen Prozesse (inkl. sich)
  - Empfängt ein Prozess eine REQ Nachricht, dann
    1. nicht im kritischen Abschnitt und will diesen auch nicht betreten $\rightarrow$ OK an Sender
    2. im kritischen Abschnitt $\rightarrow$ REQ in Queue
    3. will in kritischen Abschnitt: Zeitstempel vergleichen mit Zeitstempel des eigenen REQ (eingehende Zeit kleiner $\rightarrow$ OK an Sender, anderenfalls $\rightarrow$ REQ in Queue)
  - Prozess verlässt kritischen Abschnitt $\rightarrow$ entnimmt alle REQ aus Queue und sendet OK an die ursprünglichen Sender

Verteilter Algorithmus - 2



Verteilter Algorithmus - 3

- Vorteile
  - kein Ausfall eines Koordinators
- Nachteile
  - jetzt nicht ein Ausfallpunkte sondern n Ausfallpunkte
  - bei Einzelversendungen: jeder Prozess muss eine Liste der Gruppenmitglieder verwalten
  - schlechte Skalierbarkeit: alle Prozesse beteiligt, jeder ähnlich wie Koordinator
  - aufwändiger und langsamer Algorithmus (# der Nachrichten!)

Tokenring Algorithmus

- Prozesse werden in einem logischen Ring angeordnet
- es kreist ein Token im Ring
- besitzt ein Prozess das Token, kann dieser in den kritischen Abschnitt eintreten
- nach Verlassen des kritischen Abschnittes wird Token weitergegeben
- will kein Prozess in den kritischen Abschnitt $\rightarrow$ Token kreist

Tokenring Algorithmus - 2

- Vorteile
  - einfach
  - fair
- Nachteile
  - Ein Host (Prozess) kann abstürzen
    - Erkennung mittels Bestätigung der Übergabe oder verbindungsorientiertem Protokoll
    - Rekonfiguration des Rings notwendig

Tokenring Algorithmus - 2

- Vorteile
  - einfach
  - fair
- Nachteile
  - Ein Host (Prozess) kann abstürzen
    - Erkennung mittels Bestätigung der Übergabe oder verbindungsorientiertem Protokoll
    - Rekonfiguration des Rings notwendig, aber jeder muss gesamte Ringkonfiguration kennen!

Tokenring Algorithmus - 2

- Vorteile
  - einfach
  - fair
- Nachteile
  - Ein Host (Prozess) kann abstürzen
    - Erkennung mittels Bestätigung der Übergabe oder verbindungsorientiertem Protokoll
    - Rekonfiguration des Rings notwendig, aber jeder muss gesamte Ringkonfiguration kennen!
  - Token kann verloren gehen
    - nicht leicht zu erkennen: nach Ablauf eines Timeouts erzeugt *Monitor* neues Token
  - Doppeltes Token $\rightarrow$ Monitor
  - Monitor stürzt ab $\rightarrow$ wählen!

Wahlalgorithmus im Ring

Chang-Roberts Algorithmus

- Jeder Knoten
  - besitzt eindeutige ID (totale Ordnung gegeben)
  - hat einen Nachfolger im Ring (gerichteter Ring)
- Menge von aktiven Knoten (stellen sich der Wahl)
  - jeder sendet Nachricht mit ID p
  - empfängt Nachricht mit ID q
    - $q < p \rightarrow$ empfangene Nachricht nach `/dev/null`
    - $q > p \rightarrow p$ wird passiv und leitet q weiter
    - $q = p \rightarrow p$ hat Wahl gewonnen