

Verteilte Systeme

4. Zeit

Ansätze

Pragmatisch: Uhrensynchronisation

- Abgleich der lokalen Uhren
- Beispiele
 - Zeitabgleich nach F. Cristian
 - Berkeley-Algorithmus
 - Verteilte Synchronisation
 - Network Time Protocol (NTP)



Theoretisch: Logische Uhren

- Vergleichbarkeit kausal abhängiger Ereignisse
- Reicht in vielen Fällen aus
- Beispiele
 - Lamport-Zeit
 - Vektor-Zeit

4.1 Uhrensynchronisation

Uhrensynchronisation

Zwei Grundtechniken:

- 1 Rechner hat die exakte Uhr (z.B. Empfang des DCF77-Signals)
exaktes Nachziehen aller anderen Uhren im System
- Jeder Rechner hat eine mehr oder weniger exakte Uhr
Ständiger Uhrenvergleich und -abgleich

Grundannahme:

- Uhren haben nur eine lineare Abweichung von der Idealzeit
Für ausreichend kleine Intervalle legitim

Ziel: Vor Überschreitung der maximal tolerierbaren Abweichung synchronisieren

- Funktioniert in der Praxis bei nicht übertriebenen Anforderungen :-)
- Beispiel: Uhrensynchronisation beim Empfang eines Ethernet-Frames (Preamble)

Lineare Uhrenabweichung

Perfekte Rechneruhr tickt mit n Hz

Genauigkeit moderner Quarzuhren:

- $\sim 10^{-5}$ (< 1 Sekunde / Tag)

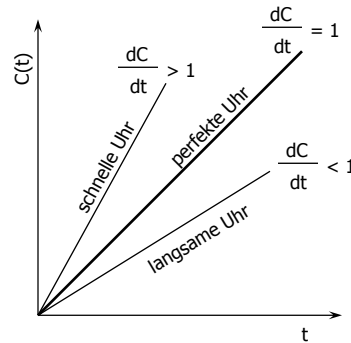
Lineare Abweichung:

$$\exists p > 0: 1 - p < \frac{dC}{dt} < 1 + p$$

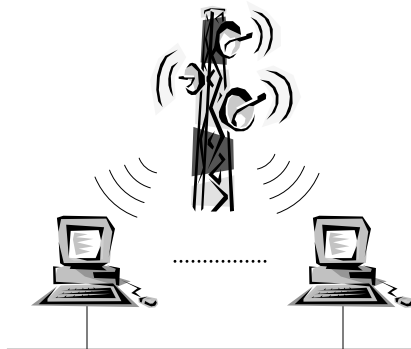
Maximale Abweichung zweier Uhren nach Δt : $2p \Delta t$

Beispiel: $p = 10^{-5}$, $\Delta t = 60s$:

- max. Abweichung 1.2 ms



DCF77-basierte Synchronisation



Zentraler Zeit-Server

- Atomuhr in Braunschweig
- Sender in Offenbach bei Frankfurt

Empfangsmodul in jedem Rechner

Maximale Drift

- Reichweite 1500 km (DCF77)
- 5 ms Differenz
- Warum nicht 10 ms?

Bedingungen für eine erfolgreiche DCF77-basierte Synchronisation?

Variante

- Zusätzliches Clock-Signal im Netz

Pech :-(-----

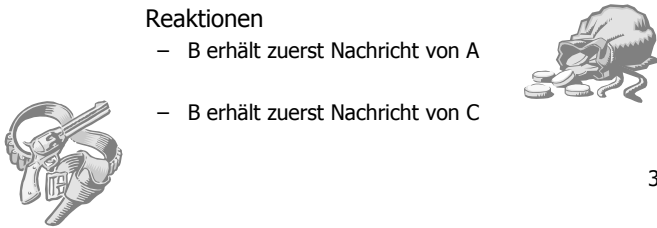
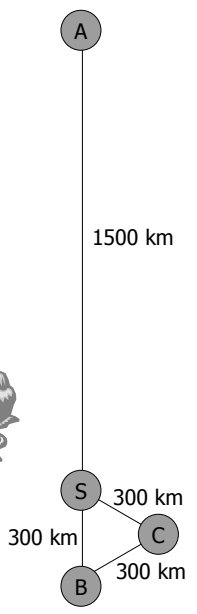
S = DCF77-Sender
GZ = Globale Zeit

Szenario:

- A sendet um 14.000 GZ an B
"Sie haben 1e10 DM gewonnen!"
- C sendet um 14.001 GZ an B
"Pokerschulden in Höhe 1e9 DM!"

Reaktionen

- B erhält zuerst Nachricht von A
- B erhält zuerst Nachricht von C

Verteilte Systeme, Wintersemester 2000/2001 Folie 4.7

Uhrenabgleich nach F. Christian -----

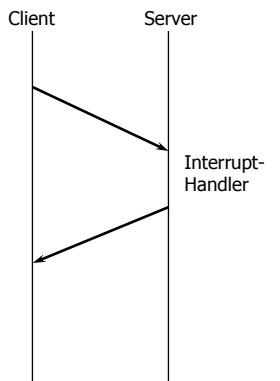
Uhrenabgleich mit passivem, zentralem Zeit-Server

Vorgabe: maximale Abweichung zwischen zwei Rechnern < a:

- Abgleichintervalle < $a/2p$

Zeit-Server sendet auf Anfrage schnellst-möglich aktuelle Uhrzeit:

- Vgl. ICMP-Nachricht (Timestamp Request)
- Übertragenen Uhrenwert übernehmen!?



Verteilte Systeme, Wintersemester 2000/2001 Folie 4.8

Zwei Probleme



Empfangener Zeitwert kleiner als lokale Uhr:

- Lokale Uhr läuft schneller
- Zeit schreitet nur voran
- Bis zum Angleich werden die Zeittakte kürzer (Anti-Schaltmillisekunden)
- Um abrupte Zeitsprünge zu vermeiden auch beim Hochsetzen (Schaltmillisekunden)

Empfangener Zeitwert berücksichtigt Transportzeit nicht

- Lokale Sende- und Empfangszeit messen
- Korrekturfaktor:

$$\frac{t_{Empfang} - t_{Senden}}{2}$$

wenn möglich lokal verbrauchte Zeit bei Server berücksichtigen
mehrere Messungen: Schnellste = geringste Verfälschung

Berkeley-Algorithmus

Aktiver, zentraler Zeit-Server

Ablauf:

- Server sendet periodisch seine Zeit an alle Rechner
- Klienten antworten mit ihrer jeweiligen Zeitverschiebung
- Server berechnet eine mittlere Zeit und teilt diese den anderen Rechnern mit
- Klienten gleichen ihre Zeiten in kleinen Schritten an
 - Schaltmillisekunden
 - Anti-Schaltmillisekunden

Network Time Protocol (NTP)

UDP-basiertes Protokoll
(Port 123)

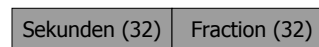
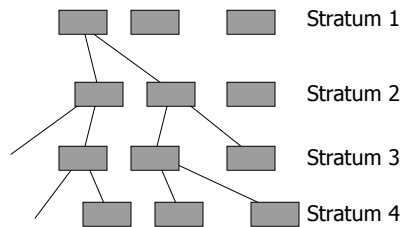
Hierarchische Struktur

- Primary Server werden z.B. über DCF77 synchronisiert
- Secondaries gleichen zur nächst höheren Ebene ab
- Minimaler Spannbaum (Bellman-Ford)
Gewicht: Stratum+Distanz

Protokoll liefert

- Clock Offset
- Roundtrip Delay
- Dispersion (max. Fehler)

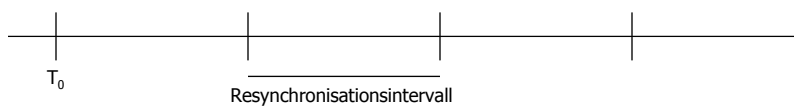
relativ zu einem Referenz-Server



Verteilte Systeme, Wintersemester 2000/2001

Folie 4.11

Verteilter Uhrenabgleich



T_0 akzeptierter gemeinsamer Zeitpunkt

Ablauf:

- Zu Beginn eines Resynchronisationsintervalls sendet jeder Rechner seine aktuelle Zeit per Broadcast
aufgrund der Drift geschieht das nicht gleichzeitig
- Einsammeln weiterer Broadcasts
- Mittelwert bilden und neue Zeit berechnen
m beste und m schlechteste Werte ignorieren (kein Abgleich mit maximal m kaputten Uhren)

Verteilte Systeme, Wintersemester 2000/2001

Folie 4.12

4.2 Logische Uhren

Kausalität

The diagram shows two vertical timelines, A and B, representing the execution of processes. Timeline A has events e1, e3, e5, e7, e9. Timeline B has events e2, e4, e6, e8, e10. Arrows indicate causal dependencies: e5 causes e6, and e8 causes e9.

Ereignisse

- Lokale Ereignisse
- Sende- und Empfangereignisse

Kausalität

- Ursache vor Wirkung
- Ereignis e_n ist kausal abhängig von e_m , wenn e_n Auswirkungen auf e_m haben kann:

$$e_n <_k e_m$$

- Kausal unabhängige Ereignisse
- Transitivität
- Partielle Ordnung

Kausale Abhängigkeit lokaler Ereignisse

$$(A, e_n) <_k (A, e_{n+l}) \quad l > 0$$

Kausalität bei Senden und Empfangen

$$(A, e_{Senden}) <_k (B, e_{Empfangen})$$

Verteilte Systeme, Wintersemester 2000/2001 Folie 4.14

Logische Uhren

Ereignissen logische Zeit zuordnen

Gesucht: Abbildung $LC : E \rightarrow H$

- E: Menge der Ereignisse mit Kausalrelation
- H: Zeitbereich (halbgeordnete Menge; "früher", "später")

Zeitstempel $LC(e)$

Uhrenbedingung

$$e_n <_k e_m \Rightarrow LC(e_n) < LC(e_m)$$

Folie 4.15

Lamportzeit

Logische Uhr LC pro Rechner

- n Bit-Zähler

Logische Uhr tickt bei Ereignissen

- Lokales Ereignis


```
LC := LC + 1;
```
- Sendeereignis


```
LC := LC + 1;
            Send(Message, LC);
```
- Empfangereignis


```
Receive(Message, LC_Sender);
            LC := Max(LC, LC_Sender) + 1;
```

Gilt die Uhrenbedingung?

Folie 4.16

Eigenschaften der Lamportzeit

Kausale Unabhängigkeit

$$LC(a) = LC(b) \Rightarrow a \parallel b$$

$$a \parallel b \Leftrightarrow \neg(a <_k b) \wedge \neg(b <_k a)$$

Umkehrung der Uhrenbedingung gilt nicht

$$LC(a) < LC(b) \not\Rightarrow a <_k b$$

$$LC(a) < LC(b) \Rightarrow (a <_k b) \vee (a \parallel b)$$

- d.h. aus den Zeitstempeln läßt sich nicht immer eindeutig festlegen, ob zwei Ereignisse kausal voneinander abhängen

Zukunft kann Vergangenheit nicht beeinflussen

$$LC(a) < LC(b) \Rightarrow \neg(b <_k a)$$

Erweiterte Lamportzeit

Lamportzeit ist nicht injektiv

$$LC(a) = LC(b) \not\Rightarrow a = b$$

Erweiterung zu einer totalen Ordnung

- Hinzufügen eines zusätzlichen Kriteriums, z.B. Rechnernummer
- Zeitstempel für Ereignis e auf Rechner A: $LC_E(A, e)$

$$LC_E(A, a) < LC_E(B, b)$$

$$\Leftrightarrow LC(a) < LC(b) \vee (LC(a) = LC(b) \wedge A < B)$$

- Alle Ereignisse haben verschiedene Zeitstempel

Erweiterte Lamportzeit kausalitätserhaltend

$$a <_k b \Rightarrow LC(a) < LC(b) \Rightarrow LC_E(A, a) < LC_E(B, b)$$

Überläufe der logischen Uhr

Überläufe praktisch ausschließen

- Abschätzung der maximalen Ereignisrate pro Rechner und der maximalen Laufzeit
- Bitanzahl von LC geeignet wählen
- Beispiel

Maximal 1000 Ereignisse pro Sekunde

32 Bit Zähler: 49 Tage, 17 Stunden, 2 Minuten, 47 Sekunden

64 Bit Zähler: ca. 584 Millionen Jahre

Spezieller Algorithmus bei Überlauf

- Auslöser: LC-Überlauf auf einem Rechner
- Lösungsansätze?



Verteilte Systeme, Wintersemester 2000/2001

Folie 4.19

Zusammenfassung Lamportzeit

Einfache Realisierung

- Verschicken zusätzlicher Information in jeder Nachricht
 - Identifikation des sendenden Rechners
 - Zeitstempel des Sendeereignisses

Uhrenbedingung erfüllt

$$a <_k b \Rightarrow LC(a) < LC(b) \Rightarrow LC_E(A, a) < LC_E(B, b)$$

Umkehrung der Uhrenbedingung gilt nicht

$$LC_E(A, a) < LC_E(B, b) \not\Rightarrow LC(a) < LC(b) \not\Rightarrow a <_k b$$

- d.h. aus den Zeitstempeln kann i.A. nicht auf eine kausale Abhängigkeit geschlossen werden

Verteilte Systeme, Wintersemester 2000/2001

Folie 4.20

Vektorzeit

Beliebige aber feste Menge von n Rechnern

- Erweiterung auf dynamische Rechnermengen möglich

Zeitstempel eines Ereignisses a ist ein Vektor der Größe n

Verschiedene Realisierungsvarianten

Schreibweisen

- Sei VC_k die logische Uhr von Rechner k
- $VC_k[j]$ bezeichne die j-te Komponente in VC_k

$$VC(a) = \begin{pmatrix} VC_0 \\ M \\ VC_{n-1} \end{pmatrix}$$

Realisierung [Mattern 1989]

Initialisierung von VC auf Rechner k

```
for (i=0; i<n; i++) VC_k[i] := 0;
```

Lokales Ereignis auf Rechner k

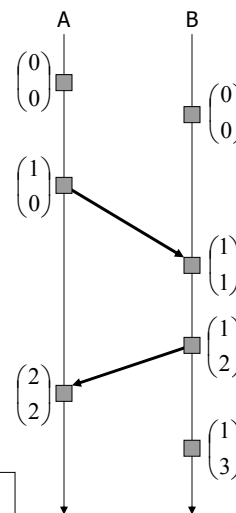
```
VC_k[k] := VC_k[k] + 1;
```

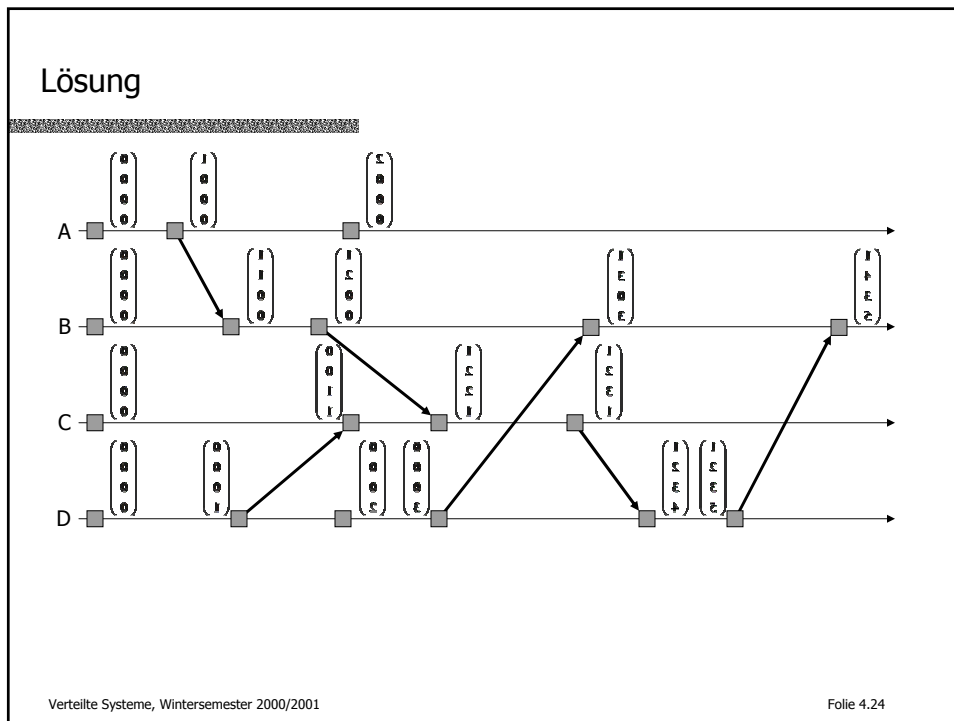
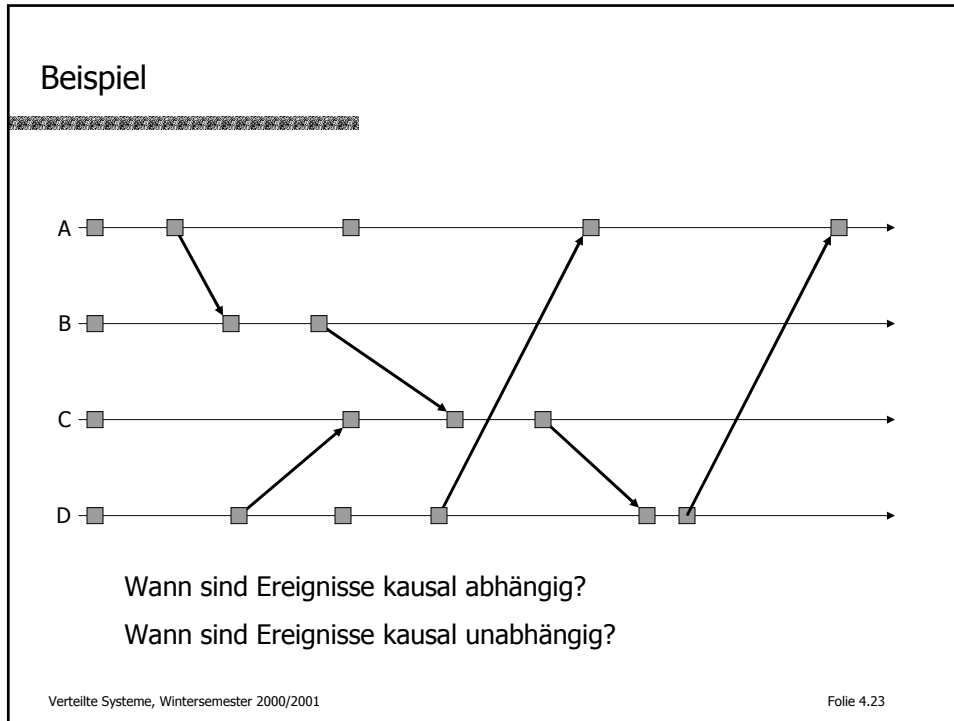
Sendereignis auf Rechner k

```
VC_k[k] := VC_k[k] + 1;
Send (Message, VC_k);
```

Empfangereignis auf Rechner k

```
VC_k[k] := VC_k[k] + 1;
Receive (Message, VC_sender);
for (i=0; i<n; i++)
    VC_k[i] := max (VC_k[i], VC_sender[i]);
```





Eigenschaften der Vektorzeit

Sei $va=VC(a)$ und $vb=VC(b)$

Definition:

Für 2 Zeitstempel $VC(a)$ und $VC(b)$ gilt:

$$va \leq vb \Leftrightarrow \forall i : va[i] \leq vb[i]$$

$$va < vb \Leftrightarrow (va \leq vb) \wedge (va \neq vb)$$

$$va \parallel vb \Leftrightarrow \neg(va < vb) \wedge \neg(vb < va)$$

Kausale Abhängigkeit

$$va \leq vb$$

Kausale Unabhängigkeit

$$va \parallel vb \Rightarrow \exists i, j \in \{0, \dots, n-1\} : (va[i] < vb[i]) \wedge (vb[j] < va[j])$$

Zusammenfassung Vektorzeit

Einfache Realisierung

- Verschicken zusätzlicher Information in jeder Nachricht
 - Identifikation des sendenden Rechners
 - Zeitvektor des Sendeereignisses

Problematik dynamischer Rechnermengen

Uhrenbedingung erfüllt

$$a <_k b \Rightarrow VC(a) < VC(b)$$

Umkehrung der Uhrenbedingung gilt ebenfalls

$$VC(a) < VC(b) \Rightarrow a <_k b$$

- d.h. aus den Zeitstempeln kann auf eine kausale Abhängigkeit geschlossen werden
- Beweis?

Zusammenfassung

Uhrensynchronisation

- Reicht in vielen Fällen meist aus
- Uhrenbedingung
 - Sanfter Abgleich einer ev. Uhrenabweichung notwendig
 - Dann gilt natürlich auch die Umkehrung

Logische Uhren

- Beschränkung auf kausal abhängige Ereignisse
- Häufig Erweiterung auf totale Ordnung
 - Kausalität bleibt weiterhin erhalten
 - "Willkürliches" Zusatzkriterium bei unabhängigen Ereignissen
- Lamportzeit ist einfachste Realisierung
 - Erfüllt Umkehrung der Uhrenbedingung nicht
- Vektorzeit
 - "Hoher" Aufwand
 - Erfüllt Umkehrung der Uhrenbedingung

Übungsaufgaben

1. Lamportzeit

Überlegen Sie sich einen Algorithmus, der die Zählerstände der logischen Uhren unabhängig von einem Überlauf reduziert. Ein solcher Algorithmus kann z.B. periodisch ausgeführt werden, um einem Überlauf vorzubeugen. Sie können von einer Beschränkung auf n beteiligte Rechner ausgehen.

2. Vektorzeit

Erweitern Sie die Vektorzeit für dynamische Rechnermengen.

- Was geschieht beim Hinzufügen eines $n+1$. Rechners?
- Was geschieht beim Austritt eines n . Rechners?

Wie kann man bei der Vektorzeit auf Zählerüberläufe reagieren?

Literatur

F. Christian

Probabilistic Clock Synchronization

Distributed Computing, Vol. 3, pp. 146-158, 1989

R. Gusella, S. Zatti

*The Accuracy of the Clock Synchronization Achieved by TEMPO
in Berkeley UNIX 4.3 BSD*

IEEE Trans. on SE, Vol. 15, pp. 847-853, 1989

(Berkeley-Algorithmus zur Uhrensynchronisation)

L. Lamport

Time, clocks and the ordering of events in a distributed system

CACM, Vol. 21, Nr. 7, pp. 558-565

(Kausalität, Vektorzeit)

F. Mattern

Verteilte Basisalgorithmen

Informatik-Fachberichte 226, Springer-Verlag, 1989

D.L. Mills

Network Time Protocol (NTP) Version 3

RFC1305