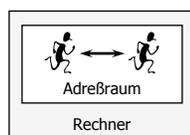


Verteilte Systeme

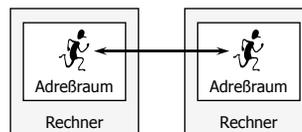
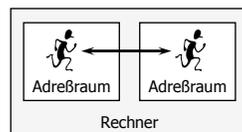
19. Distributed Shared Memory

Sharing! No Sharing! Sharing?



Evolution der Berechnungsmodelle

- Vergangenheit
- Gemeinsamer Speicher
 - Einzelrechner

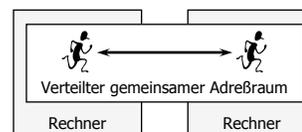


Gegenwart

- Nachrichtenkommunikation
- Verteilte Systeme

Zukunft?

- Verteilte Systeme
- Gemeinsamer Speicher



Modell

Physischer Adreßraum des Knoten

- Lokal adressierbarer Speicher
- Verteilter Speicherbereich

Kommunikation durch Lese- und Schreiboperationen

Speicherbasierte Synchronisationsprimitive möglich

- Semaphore, etc.

Unterschiedliche Granulate

- Seiten
- Objekte

LINDA

- Variablen

Shared Variables

Verteilte Systeme, Wintersemester 2000/2001
Folie 19.3

Realisierungsidee

Referenz im gemeinsamen Adreßbereich

„Speicher vorhanden“

Zugriff

„Speicher nicht vorhanden“

Seitenfehler

Aufenthaltort des referenzierten Speichers ermitteln

Speicherbereich anfordern

- Nachricht

Bereich empfangen

- Nachricht

Zugriff

Copy on Reference

Verteilte Systeme, Wintersemester 2000/2001
Folie 19.4

Einfache Variante

Genau eine Kopie jeder Seite

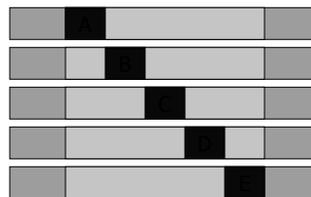
Jede Seite hat genau einen Besitzer

- Partitionierung des Adreßbereichs

Algorithmus?

Diskussion?

- Vorteile
- Nachteile



Verteilte Systeme, Wintersemester 2000/2001

Folie 19.5

Realisierung

Client A (Anforderung):

```
Seitenfehler (Adresse x) {
  Ermittle Seite p aus x;
  Bestimme Owner o;
  Send Request(p) to o;
  Receive(Content(p));
  Seitentabelle aktualisieren;
}
```

Client B (Im Besitz von p):

```
Receive Forward(p,X) from Z {
  Send Content(p) to X;
  Seitentabelle aktualisieren;
}
```

Owner:

```
init {
  user(p) = self;
}

Receive Request(p) from X {
  if (user(p) == self) {
    Send(Content(p)) to X;
    Seitentabelle korrigieren;
  }
  else {
    // user(p) == Y;
    Send Forward(p,X) to Y;
    user(p) = X;
  }
}
```

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.6

Vor- und Nachteile

Vorteile

Einfache Implementierung
Max. 3 Nachrichten pro
Seitenfehler

Nachteile

Einschränkung der Parallelität
– Nur einer kann auf Seite
zugreifen
Gefahr des Seitenflatterns über
Rechnergrenzen
– Zwei Klienten referenzieren
ständig dieselbe Seite

Realisierung 2

Mehrere Lesekopien der Seite, Maximal eine Schreibkopie

Verwaltung analog über Zuordnung Seite-Besitzer

- Partitionierung des Adreßbereich

Algorithmus?

Vor- und Nachteile?

Realisierungsansatz

Seitenreplikate

- Client-Menge pro Seite
- Verwaltung z.B. durch Owner
- Weiterleitung einer Anfrage an ein Mitglied der Client-Menge
- Owner aktualisiert Menge

Problematik: Schreibenforderung auf Seite

- Lesekopien werden ungültig
- 2 Ansätze
 - Invalidierung innerhalb der Client-Menge
 - Einzelnachrichten
 - ggf. Broadcast
 - Gültigkeit vor jedem Zugriff prüfen (vgl. WWW)

Problematik: Schreibgranulat

- Zeitpunkt für erneute Lesekopien?

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.9

Vor- und Nachteile

Vorteile

- Geringere Einschränkung der Parallelität
 - Gleichzeitiger lesender Zugriff
- Lesezugriff
 - max. 3 Nachrichten

Nachteile

- Gefahr des Seitenflatterns über Rechengrenzen
 - Zwei Klienten referenzieren ständig dieselbe Seite
- Schreibzugriff
 - Nachrichtenaufwand für Invalidierung bzw. Validierung

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.10

Weitere Verbesserungen?



Verteilte Systeme, Wintersemester 2000/2001

Folie 19.11

Realisierung 3

Mehrere Lese- und Schreibkopien der Seite

Kritischer Fall

- 2 und mehr „gleichzeitige“ Schreibzugriffe
- Wann übernimmt die Seite welchen Wert?

Algorithmus?

Vor- und Nachteile?

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.12

Konsistenzmodelle

Mehrere Replikate

- Vorteil: Bessere Performance
- Nachteil: Wahrung der Speicherkonsistenz

Gilt für Multiprozessoren und verteilte Systeme

Implizite Annahme: Strikte Konsistenz

- Zugriff auf Speicherzelle X liefert den zuletzt zugewiesenen Wert (Most recent)
- „Zuletzt“ im verteilten Fall nicht eindeutig identifizierbar

Abschwächungen des Konsistenzbegriffs

- Wohldefiniert „seltsames“ Verhalten aus Sicht der Anwendung
- Ggf. anwendungsspezifische Mechanismen zur Einhaltung stärkerer Konsistenzkriterien notwendig

Sequentielle Konsistenz

Definition

- Prozeß sieht eine Folge von Lese- und Schreibzugriffen auf eine Speicherzelle
- Die sichtbare Folge muß **einer** sequentiellen Abarbeitung aller Lese- und Schreiboperationen entsprechen
- Alle Prozesse sehen dieselbe Folge

Bemerkungen

- Definition unabhängig von einem Zeitbegriff
- Nichtdeterminismus
- Performance-Probleme

Leseoptimierung \Rightarrow
Schreiben teuer

Schreiboptimierung \Rightarrow
Lesen teuer

P1: $W(x)=1$
P2: $R(x)=0 \quad R(x)=1$

P1: $W(x)=1$
P2: $R(x)=1 \quad R(x)=1$

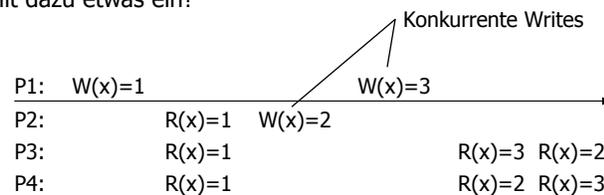
Kausale Konsistenz

Definition

- Potentiell kausal abhängige Schreibzugriffe werden von allen beteiligten Prozessen in derselben Reihenfolge wahrgenommen
- Konkurrente Schreibzugriffe können von verschiedenen Prozessen in unterschiedlicher Reihenfolge beobachtet werden

Realisierungsansatz

- Protokoll: Welcher Prozeß sieht wann welche Writes?
- Wem fällt dazu etwas ein?



PRAM-Konsistenz oder Prozessorkonsistenz

Definition

- Nur die Schreibzugriffe eines Prozesses (Rechners) erscheinen für alle beteiligten Prozesse in derselben Ordnung

Einfache Realisierbarkeit

- Wie?

PRAM = Pipelined RAM

Spezieller Multiprozessor (Gegenstand der Forschung). Schreibzugriffe eines Prozessors werden in einer Pipeline zu den Speicherelementen geführt. Man erhofft sich eine höhere Performance trotz beschränkter Speicherbandbreite. PRAMs sind technisch noch nicht sinnvoll realisierbar.

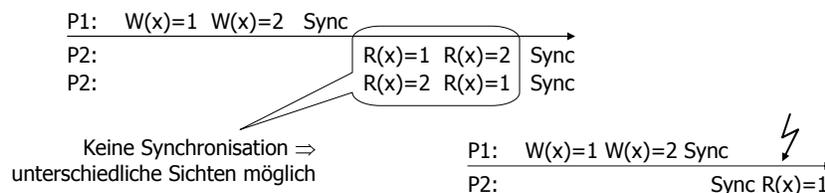
Schwache Konsistenz

Definition

- Zugriff zu Synchronisationsvariablen sind sequentiell konsistent
- Kein Zugriff auf Synchronisationsvariablen, solange nicht alle ausstehenden Schreibzugriffe abgeschlossen sind
- Keine lesenden oder schreibenden Datenzugriffe, solange nicht alle Zugriffe auf Synchronisationsvariablen abgeschlossen sind

Bemerkungen

- Konsistenz nur bei ausgezeichneten Synchronisationsvariablen



Verteilte Systeme, Wintersemester 2000/2001

Folie 19.17

Release-Konsistenz

Zusätzliche Operationen

- ACQUIRE: Beginn eines (bzgl. der Speicherkonsistenz) kritischen Abschnitts
- RELEASE: Ende eines (bzgl. der Speicherkonsistenz) kritischen Abschnitts

Definition

- Vor dem Zugriff auf eine Speicherzelle müssen alle ACQUIRE-Operationen des zugreifenden Prozesses erfolgreich beendet sein
- Eine RELEASE-Operation ist beendet, wenn alle Lese- und Schreibzugriffe des Prozesses beendet sind
- Die Folge der ACQUIRE- und RELEASE-Operationen ist prozessor-konsistent

Compiler oder Programmierer müssen ACQUIRE- und RELEASE-Operationen explizit angeben

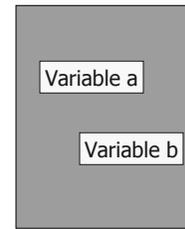
Verteilte Systeme, Wintersemester 2000/2001

Folie 19.18

Bemerkungen DSM

Viele weitere Detailprobleme

- z.B. False Sharing
 - Mehrere Datenstrukturen in einer Seite
 - Austausch nur auf Seitengranulat
 - Schutzmechanismen nur Seitenbasiert



Seite X

Kritisches Laufzeitverhalten

- Performance abhängig von Zugriffsmuster und Variablenplatzierung
- Nicht-Determinismus

Primär noch Forschungsgegenstand

Methodische Kritik

- Vordergründig ansprechendes Programmiermodell
- Nachrichtenaufwand verborgen und damit unberechenbar
- „Nicht verteilt gedacht“

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.19

The END

Ernüchterung nach Euphorie der ersten Jahre

- Leistungssteigerung: Meist nur durch erneutes Programmieren
- Fehlertoleranz: Enorme Materialschlacht
- Natürliche Verteilung: Mensch denkt eher sequentiell

Inhärente Probleme

- Keine globale Zeit
- Kein globaler Zustand
- Indeterminismus
- Ausfälle
- ...



Reizvoll

Verteilte Systeme trotz aller Widrigkeiten extrem wichtig

- WWW, E-Commerce, Multiplayer, ...

Verteilte Systeme, Wintersemester 2000/2001

Folie 19.20