Überblick

Problemstellung

- Mehrere Prozesse greifen auf gemeinsame Daten/Ressourcen zu
- Zugriff muss koordiniert werden: Immer nur ein Prozess darf den kritischen Abschnitt (KA) betreten

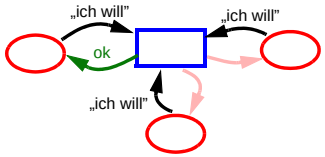
Literatur

- Original-Publikationen zu den einzelnen Algorithmen
- Überblick über die meisten relevanten Forschungsarbeiten

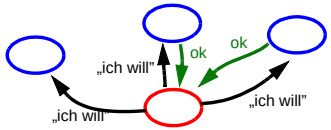
[Cao et al.] G. Cao, M. Singhal
A Delay-Optimal Quorum-Based Mutual Exclusion Algorithm for Distributed Systems
IEEE Transactions on Parallel and Distributed Systems, 12(12):1256–1268, 2001.

[Singhal et al.] M. Singhal, N. Shivaratri
Advanced Concepts in Operating Systems
McGraw-Hill, Inc., 1994.

Zentraler Koordinator (→ Alle fragen einen)



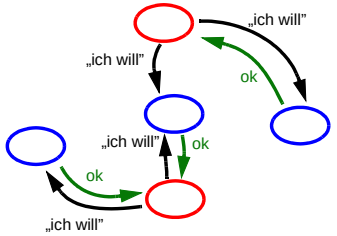
Vollständig verteilt (→ Jeder fragt jeden)



Beispiel: Lock-Protokoll von Lamport (siehe Übung)

Quorenbasierte Algorithmen

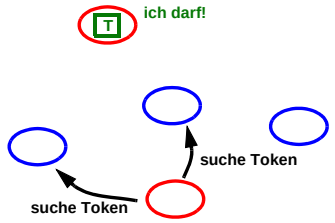
- Dezentral, aber reduzierter Aufwand
- *Jeder fragt ausreichend viele*



- Beispiele
  - Maekawa: Quorenbasierter gegenseitiger Ausschluss
  - Sanders: Universeller erlaubnisbasierter Algorithmus
  - Cao und Singhal: Optimierte Koordinierungsverzögerung

Tokenbasierte Algorithmen

- Wer Token besitzt, der darf in den kritischen Abschnitt!
- Wie bekommt man das Token?



- Beispiele
  - Perpetuum-Mobile-Algorithmus: Token wandert alleine
  - Algorithmus von Suzuki und Kasami: Broadcast-Suche
  - Algorithmus von Raymond: Suche auf Baumstrukturen
  - Algorithmus von Singhal: „Intelligenter“ Broadcast

Formale Anforderungen

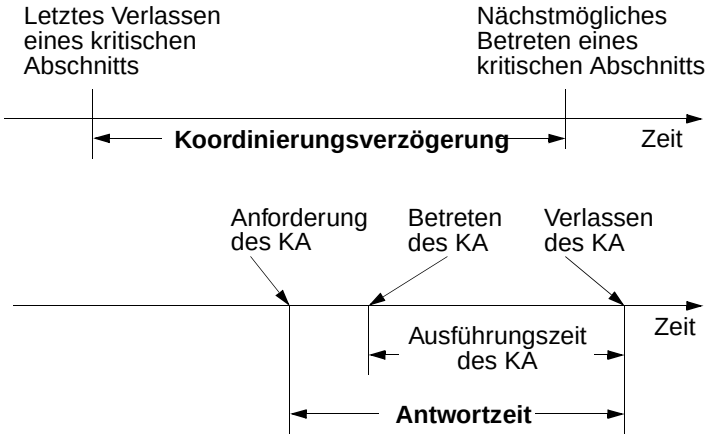
- **Gegenseitiger Ausschluss:** Zu keinem Zeitpunkt hat mehr als ein Prozess die Erlaubnis, den kritischen Abschnitt auszuführen
- **Lebendigkeit (bzw. Verklemmungsfreiheit):** Wenn sich kein Prozess im kritischen Abschnitt befindet und mindestens ein Prozess diesen betreten möchte, erhält ein Prozess in endlicher Zeit die Erlaubnis, den kritischen Abschnitt zu betreten

Strengere Anforderung für Lebendigkeit

- **Aushungerungsfreiheit:** Ein Prozess, der den kritischen Abschnitt betreten möchte, erhält innerhalb endlicher Zeit die Erlaubnis dazu

Weitere wesentliche Gesichtspunkte

- Fairness
  - Die Zulassung erfolgt in der Reihenfolge der Anforderungen
  - Eine globale Ordnung nach physikalischer Zeit ist evtl. nicht möglich. Üblich daher „Reihenfolge“ gemäß Ordnung durch logische Uhren.
  - Fairness impliziert Aushungerungsfreiheit (aber nicht umgekehrt)
- Fehlertoleranz
- Erzeugte Netzlast (Nachrichtenanzahl)
- Erzielte Performanz

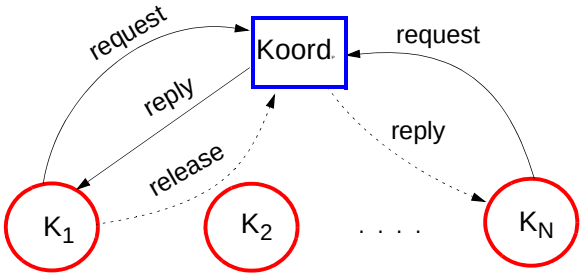


Durchsatz: Anzahl KA pro Zeit; 1/(Koord.Verz.+Ausführungszeit)

Zentraler Koordinator

Nachrichtentypen

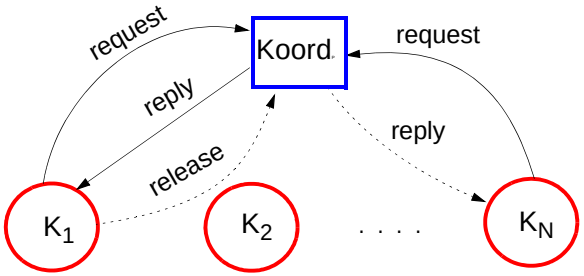
- `REQUEST` Anforderung des kritischen Abschnitts
- `REPLY` Zuteilung des kritischen Abschnitts
- `RELEASE` Freigabe des kritischen Abschnitts



Zentraler Koordinator

Eigenschaften

- Anzahl der Nachrichten: 3 pro kritischem Abschnitt
- Koordinierungsverzögerung: 2 Nachrichtenlaufzeiten
  - `RELEASE`
  - `REPLY`



Quorenbasierte Algorithmen

Überlegung zur Reduktion der Nachrichtenanzahl

- Lamport-Algorithmus (siehe 6. Übungsaufgabe) benötigt $O(N)$ Nachrichten pro kritischem Abschnitt
- Quorenbasierte Algorithmen verfolgen die Idee, nur mit einer Teilmenge aller Knoten (einem Quorum) zu interagieren, um den kritischen Abschnitt betreten zu können
- Quorum-Mengen sollen so gebildet werden, dass
  - der gegenseitige Ausschluss sichergestellt ist
  - der Kommunikationsaufwand möglichst weit reduziert wird
  - eine möglichst gleichmäßige Verteilung der Last erreicht wird

Quorenbasierte Algorithmen

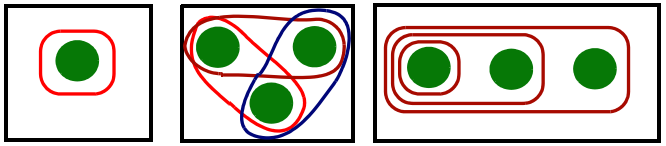
Allgemeine Definition von „Quorensystem“

Menge von Knotenmengen („Quoren“), von denen sich je zwei in mindestens einem Element überschneiden

Vielfältige Anwendungsgebiete in verteilten Systemen

- Gegenseitiger Ausschluss
- Fehlertolerante Einigungs- und Commit-Protokolle
- Gemeinsame verteilte Variablen
- ...

Beispiele



Quorenbasierte Algorithmen

Nicht alle Quorensysteme sind „gleich gut“

- Evtl. unnötiger Aufwand bei Überschneidung in mehreren Knoten
- Ungleiche Lastverteilung
  - falls manche Knoten häufiger in Quoren verwendet werden als andere
  - bei Fehlertoleranz-Anwendungen: Ausfall mancher Knoten kann mehr Schaden anrichten als der Ausfall von anderen Knoten

Satz (hier nicht bewiesen)

- Ein ideales Quorensystem (alle Quoren sind gleich groß und überschneiden sich in nur einem Element) existiert nur, wenn sich die Knotenanzahl N darstellen lässt in der Form $N = p^m(p^m + 1) + 1$, wobei p eine Primzahl ist.
- Beispiel: $N = 13 = 3^1(3^1 + 1) + 1$

Quorenbasierte Algorithmen

Einfache Quorensysteme

- Spezialfälle
  - Ein zentraler Koordinator
  - Alle Knoten in einem Quorum
- Mehrheitsquoren: Alle Mengen von Knoten mit mehr als der Hälfte aller Knoten
- Gewichtete Mehrheitsquoren
  - Jeder Knoten K_i hat Gewicht G_i
  - Gesamtgewicht aller Knoten $G_{ges} = \sum_i G_i$
  - Quoren sind alle Knotenmengen mit Gesamtgewicht $> G_{ges}/2$

Quorenbasierte Algorithmen

Quadratgitterkonstruktion von Quoren

- Eintragen der Knoten-IDs in ein Quadratgitter
- Quorum
  - Alle Elemente in derselben Zeile und Spalte wie der Knoten selbst
  - Die Quoren überschneiden sich in der Regel in zwei Elementen
- Quorengröße ist $2\lceil\sqrt{N}\rceil - 1$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Algorithmus von Maekawa

Überblick

- Verteilter Algorithmus, der Quoren zur Reduktion des Kommunikationsaufwands verwendet
- Logische Uhren nach Lamport und Knoten-IDs werden verwendet, um eine totale Ordnung auf Anforderungen zu definieren

Datenstrukturen eines Knotens P_i

- **Anforderungsmenge**: Menge von Knoten, von denen P_i eine Anforderung für den kritischen Abschnitt empfangen hat
- **Erlaubnismenge**: Menge von positiven Antworten anderer Knoten auf eine eigene Anforderung für den kritischen Abschnitt
- **Belegungskennung**: „*belegt für P_j*“, falls P_j von P_i die Erlaubnis für den kritischen Abschnitt erhalten hat, sonst „*frei*“

Algorithmus von Maekawa

Grundidee des Algorithmus

- Knoten P_i fordert kritischen Abschnitt an: `REQUEST(i, t)` an alle Knoten des Quorums von P_i
- P_j empfängt `REQUEST(i, t)`: P_i in Anforderungsmenge aufnehmen, weitere Aktionen abhängig vom Zustand der P_j-Belegungskennung
  - *frei*
    - Wechsel in den Zustand „belegt für P_i“
    - Senden einer `LOCKED`-Nachricht an P_i
  - *belegt für P_r*
    - Senden von `LOCKED` wird verzögert
    - Weitere Maßnahmen: siehe später
- P_i empfängt `LOCKED` von P_j
  - P_j in die Erlaubsnismenge aufnehmen
  - Falls alle Quorumsmitglieder in der Erlaubnismenge sind, darf der kritische Abschnitt betreten werden

Algorithmus von Maekawa

Grundidee des Algorithmus (2)

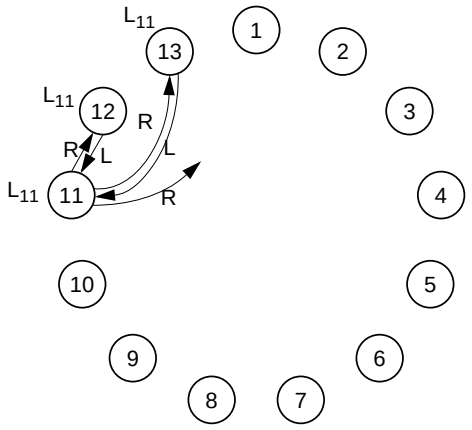
- Knoten P_i verlässt den kritischen Abschnitt
  - Erlaubnismenge leeren
  - `RELEASE`-Nachricht an alle Quorumsmitglieder
- Knoten P_j empfängt `RELEASE`-Nachricht von P_i
  - Entfernen von P_i aus Anforderungsmenge
  - Falls die Anforderungsmenge weitere Anforderungen enthält: Für älteste Anforderung von P_r aus der Anforderungsmenge
    - Wechsel in den Zustand „belegt für P_r“
    - Senden einer `LOCKED`-Nachricht an P_r

Zwischenstand

- Bisheriger Algorithmus gewährleistet den gegenseitigen Ausschluss
- Es kann allerdings zu **Verklemmungen** kommen!

Algorithmus von Maekawa

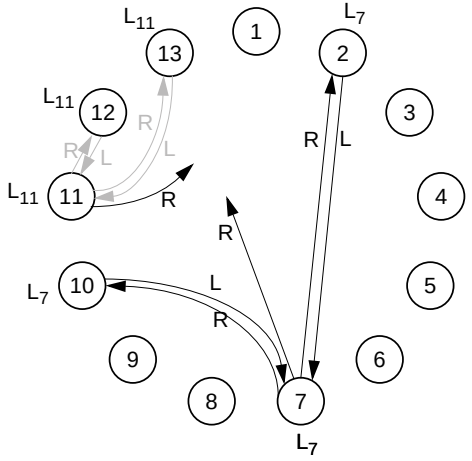
Beispiel-Ablauf (1)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

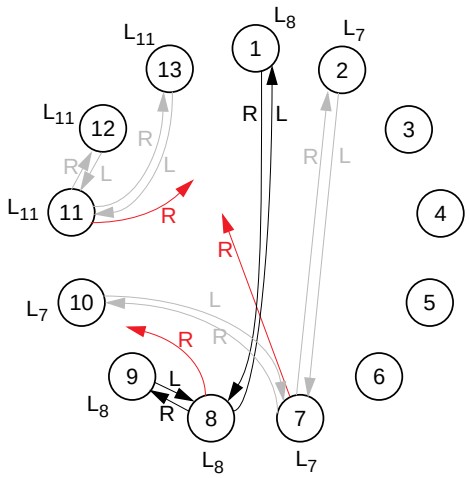
Beispiel-Ablauf (2)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

Beispiel-Ablauf (3)



S_1 = { 1, 2, 3, 4 }
S_2 = { 2, 5, 8,11 }
S_3 = { 3, 6, 8,13 }
S_4 = { 4, 6,10,11 }
S_5 = { 1, 5, 6, 7 }
S_6 = { 2, 6, 9,12 }
S_7 = { 2, 7,10,13 }
S_8 = { 1, 8, 9,10 }
S_9 = { 3, 7, 9,11 }
S_{10} = { 3, 5,10,12 }
S_{11} = { 1,11,12,13 }
S_{12} = { 4, 7, 8,12 }
S_{13} = { 4, 5, 9,13 }

Algorithmus von Maekawa

Maßnahmen zum Erkennen und Auflösen von Verklemmungen

- Im Konfliktfall bekommt eine Anforderung mit kleinerem Zeitstempel Vorrang
- Ausnahme: eine Anforderung mit größerem Zeitstempel hat bereits die erforderliche Erlaubnis für den kritischen Abschnitt erhalten

Einführung von drei weiteren Nachrichtentypen

- `FAILED`: Zeigt an, dass die Belegungskennung eines Knotens derzeit eine Anforderung mit kleinerem Zeitstempel aufweist
- `INQUIRE`: Bittet um die Rückgabe einer Erlaubnis (`LOCKED`)
- `RELINQUISH`: Gibt Erlaubnis zurück

Algorithmus von Maekawa

Erweiterungen zur Deadlock-Behandlung

- P_j empfängt `REQUEST(i, t)` von P_i, hat P_i in die Anforderungsmenge aufgenommen, und ist „belegt für P_r“
  - Falls P_i ältestes Element in der Anforderungsmenge: P_j sendet `INQUIRE`-Nachricht an P_r
  - Falls nicht: P_j sendet `FAILED` an P_i
- P_r empfängt `INQUIRE`-Nachricht von P_j
  - Falls P_r bereits `FAILED` empfangen hat
    - Senden einer `RELINQUISH`-Nachricht an P_j
    - Entfernen von P_j aus der Erlaubnismenge
  - Ansonsten: Empfang von `INQUIRE` merken (bei nachfolgendem Empfang von `FAILED` muss das Senden von `RELINQUISH` nachgeholt werden)

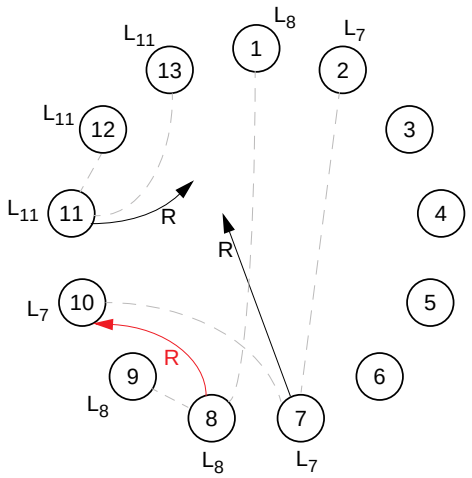
Algorithmus von Maekawa

Erweiterungen zur Deadlock-Behandlung (2)

- Wenn P_j die Nachricht `RELINQUISH` von P_i empfängt
  - Belegung für P_i aufheben
  - Für ältestes Element P_r in der Anforderungsmenge: Wechsel des Zustands der Belegungskennung zu „belegt für P_r“
  - Senden einer `LOCKED`-Nachricht an P_r
- Wenn P_i von P_j eine `FAILED`-Nachricht empfängt
  - Falls P_i bereits `INQUIRE` von einem Knoten P_k empfangen hat
    - Senden einer `RELINQUISH`-Nachricht an P_k
    - P_k aus Erlaubnismenge entfernen
  - Ansonsten: Empfang von `FAILED` merken (bei nachfolgendem Empfang von `INQUIRE` muss das Senden von `RELINQUISH` nachgeholt werden)

Algorithmus von Maekawa

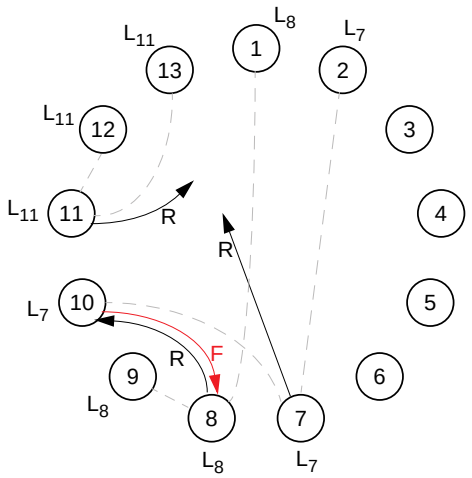
Beispiel-Ablauf (4)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

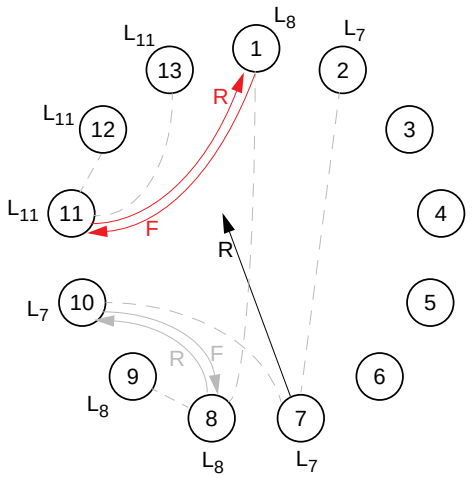
Beispiel-Ablauf (5)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

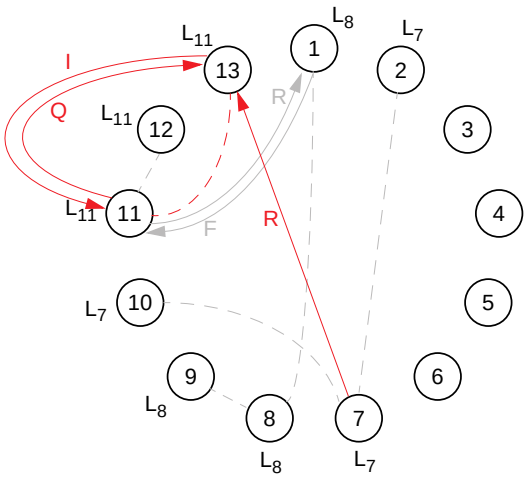
Beispiel-Ablauf (6)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

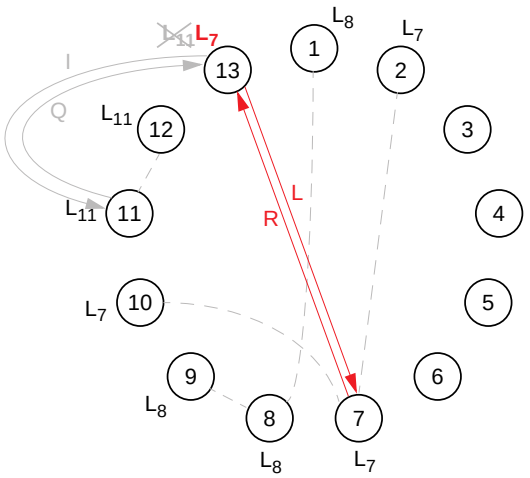
Beispiel-Ablauf (7)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

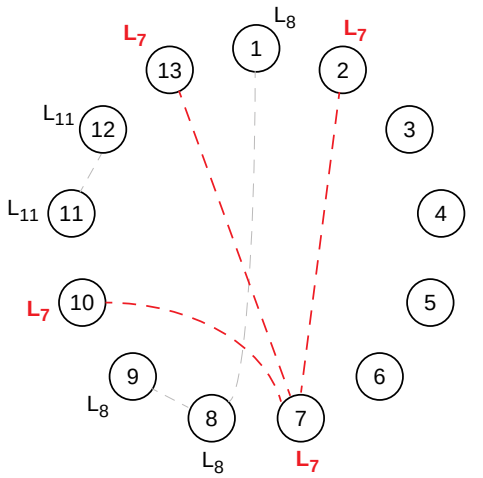
Beispiel-Ablauf (8)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

Beispiel-Ablauf (9)



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8,11 \}$
$S_3 = \{ 3, 6, 8,13 \}$
$S_4 = \{ 4, 6,10,11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9,12 \}$
$S_7 = \{ 2, 7,10,13 \}$
$S_8 = \{ 1, 8, 9,10 \}$
$S_9 = \{ 3, 7, 9,11 \}$
$S_{10} = \{ 3, 5,10,12 \}$
$S_{11} = \{ 1,11,12,13 \}$
$S_{12} = \{ 4, 7, 8,12 \}$
$S_{13} = \{ 4, 5, 9,13 \}$

Algorithmus von Maekawa

Bezeichnungen

- T: mittlere Nachrichtenlaufzeit
- E: mittlere Ausführungszeit für kritischen Abschnitt
- K: mittlere Größe eines Quorums

Eigenschaften des Algorithmus

- Antwortzeit: $2T + E$
- Koordinierungsverzögerung: $2T$
- Nachrichten
  - keine Konflikte: $3(K-1)$ pro kritischem Abschnitt
  - Konfliktfall, normal: $4(K-1)$ pro kritischem Abschnitt `[FAILED]`
  - ungünstigster Fall: $5(K-1)$ pro kritischem Abschnitt `[INQUIRE, RELINQUISH]`

$\Rightarrow O(\sqrt{N})$ Nachrichten bei geeigneten Quoren

Tokenbasierte Algorithmen

Prinzip

- Im System existiert genau ein Token
- Ein Prozess kann einen kritischen Abschnitt nur betreten, wenn er über das Token verfügt

Passive Verfahren

- „Perpetuum Mobile“

Aktive Verfahren

- Algorithmus von Suzuki/Kasami: Vollständige Suche im ganzen Netz
- Algorithmus von Raymonds: Gezielte Suche nach vordefinierter Baumstruktur
- Algorithmus von Singhal: Reduktion der Nachrichtenzahl durch intelligente, selbstadaptierende Suche nach dem Token

Perpetuum Mobile

Vorgehen

- Token wird automatisch auf (virtuellem) Ring weitergeschickt
- Knoten, der kritischen Abschnitt betreten will, wartet auf Token

Eigenschaften

- Optimalfall
  - (Fast) alle Knoten wollen (fast) immer in den kritischen Abschnitt
  - (Meist) nur eine Nachricht zwischen allen kritischen Abschnitten

  → Nahezu optimal
- Normalfall
  - Durchschnittliche Wartezeit auf Token: $N/2$ Nachrichtenlaufzeiten
  - Unbeschränkte Anzahl von Nachrichten, selbst wenn kein Knoten den kritischen Abschnitt betreten will

Aktive tokenbasierte Algorithmen

Generelles Prinzip

- Ein Prozess, der in seinen kritischen Abschnitt eintreten will, aber nicht über das Token verfügt, versendet `REQUEST`-Anforderungen
- Ein Prozess, der `REQUEST`-Anforderungen erhält, übergibt das Token, sobald es frei ist, an einen anfordernden Knoten

Problemstellungen

- Erzeugung genau eines Tokens
- Bestimmung des Prozesses, der den Token erhält
- Welche `REQUEST`-Anforderungen sind veraltet, welche aktuell?

Algorithmus von Suzuki und Kasami

Datenstrukturen

- Pro Prozess (lokal): Sequenznrn. bekannter Anfragen (`int RN[n]`)
- Datenstruktur des Tokens
  - `int LN[n]`: `LN[i]` enthält Sequenznr. des letzten KA-Eintritts von P_i
  - `IntFifo queue`: Warteschlange für Knoten, die das Token wollen

Anfordern des Tokens

- Wenn Prozess P_i das Token benötigt, erhöht er seine Sequenznummer `RN[i]` und sendet eine Nachricht `REQUEST(i,RN[i])` an alle anderen Knoten
- Wenn ein Prozess P_j eine Nachricht `REQUEST(i,sn)` empfängt, setzt er `RN[i]=max(RN[i],sn)`
- Falls P_j über das Token verfügt und nicht im kritischen Abschnitt ist, sendet er es an P_i, wenn `RN[i]==LN[i]+1` ist

Algorithmus von Suzuki und Kasami

Ausführen eines kritischen Abschnitts

Prozess P_i führt den kritischen Abschnitt aus, sobald er über das Token verfügt

Verlassen eines kritischen Abschnitts

- `LN[i]:=RN[i]`
- Jeder Prozess P_j, der nicht in `queue` vermerkt ist und für den gilt, dass `RN[j]==LN[j]+1` ist, wird an `queue` angefügt
- Falls `queue` nicht leer ist, wird der erste Prozess aus `queue` entfernt und das Token an ihn verschickt

Algorithmus von Suzuki und Kasami

Bezeichnungen

- T: mittlere Nachrichtenlaufzeit
- N: Anzahl der Knoten
- E: mittlere Ausführungszeit für kritischen Abschnitt

Eigenschaften des Algorithmus

- Antwortzeit: $2T + E$
- Nachrichten: N pro kritischem Abschnitt
- Koordinierungsverzögerung: T

Fehlertoleranz

- Knoten, die `REQUEST`s ausgesendet haben, dürfen nicht ausfallen
- Knoten, der das Token besitzt, darf nicht ausfallen

Gegenseitiger Ausschluss

Zusammenfassung

Bewertungskriterien für die verschiedenen Algorithmen