Aufgabe 6: Algorithmen für gegenseitigen Ausschluß in verteilten Systemen (Maekawa, Sanders, tokenbasierte Ansätze, Vergleich der Algorithmen)

Literatur:

[1] Singhal, M.; Shivaratri, N. G.
Advanced Concepts in Operating Systems
McGraw-Hill, Inc: New York et al. 1994

a) Beschreiben Sie kurz den Algorithmus von Maekawa! Wie werden potentielle Deadlocks behandelt?

b) Was repräsentiert der Algorithmus von Sanders und worin liegt sein entscheidender Vorteil?

c) Beschreiben Sie kurz folgende tokenbasierte Algorithmen zum gegenseitigen Ausschluß in verteilten Systemen:
   - trivialer Algorithmus via "Perpetuum Mobile"
   - Suzuki/Kasami
   - Singhal
   - Raymond

d) Stellen Sie Kriterien auf, nach denen die bisher untersuchten Verfahren zum gegenseitigen Ausschluß in verteilten Systemen beurteilt werden können!

e) Wenden Sie die von Ihnen aufgestellten Kriterien auf die Algorithmen an!

Lösungsvorschlag Aufgabe 6

a) Algorithmus von Maekawa - Beschreibung

Die Menge der Knoten wird in sich überlappende Teilmengen gegliedert, in die sogenannten Quoren. Der Durchschnitt je zweier Teilmengen darf nicht leer sein. Idealerweise sind diese alle gleich groß ($|S_i|=K$) und jeder Knoten kommt in gleich vielen Teilmengen vor. Der optimale Fall liegt dann vor, wenn die Größe des Quorums $K \approx \sqrt{N}$ ist. Jedem Prozeß ist sein eigenes Quorum zugeordnet. Es wird eine totale Ordnung durch eine logische Zeit vorausgesetzt.

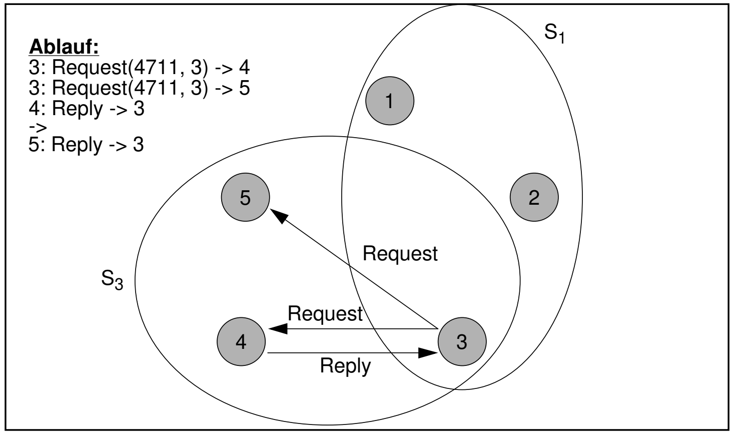
Ein Prozeß darf seinen kritischen Abschnitt betreten, wenn er die Erlaubnis von allen Mitgliedern seines Quorums hat. Um Verklemmungen zu vermeiden kann eine bereits erteilte Erlaubnis unter bestimmten Voraussetzungen (noch nicht im kritischen Abschnitt) wieder zurückgenommen werden.

Anzahl der zu sendenen Nachrichten: K-1 Anforderungen an jeden Knoten im Quorum, K-1 Antworten (in etwa nach dem Algorithmus von Ricart & Agrawala) und K-1 Freigabenachrichten. Zusätzlich können 2(K-1) Nachrichten zur Verklemmungsvermeidung notwendig sein.

Beispiel:

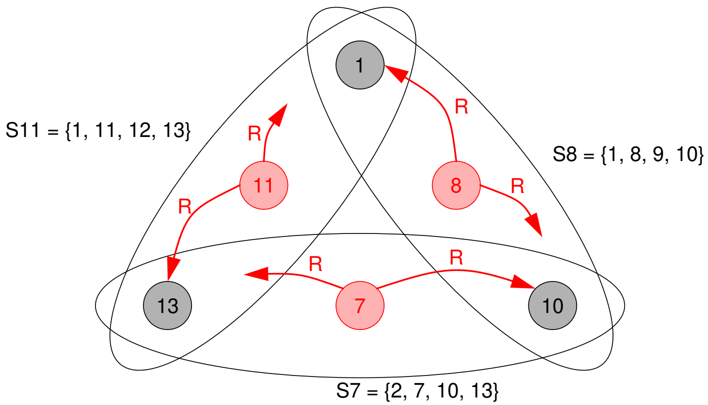
Die Anzahl der Knoten sei N=5. D.h. N entspricht nicht der Bedingung für die ideale Größe und damit gibt es keine einheitliche Quorumgröße K und die Anzahl der zu sendenden Nachrichten ist im Durchschnitt größer als $\sqrt{N}$:

$S_1 = \{ 1, 2, 3 \}$	(6 N)	Durchschn. Anzahl Nachrichten:
$S_2 = \{ 2, 4 \}$	(3 N)	
$S_3 = \{ 3, 4, 5 \}$	(6 N)	im optimalen Fall: $(\sqrt{5} - 1) \times 3 = 3.7$
$S_4 = \{ 1, 4, 5 \}$	(6 N)	in unserem Beispiel: 4.8
$S_5 = \{ 2, 5 \}$	(3 N)	

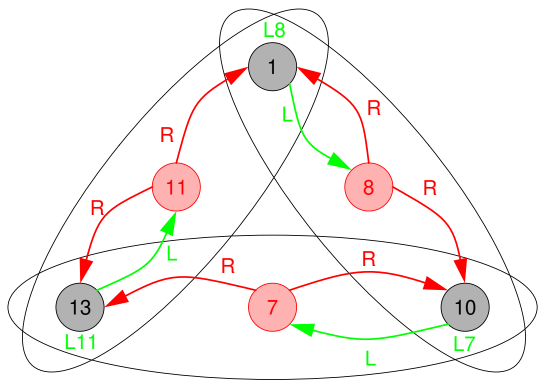


Deadlockbehandlung beim Algorithmus von Maekawa

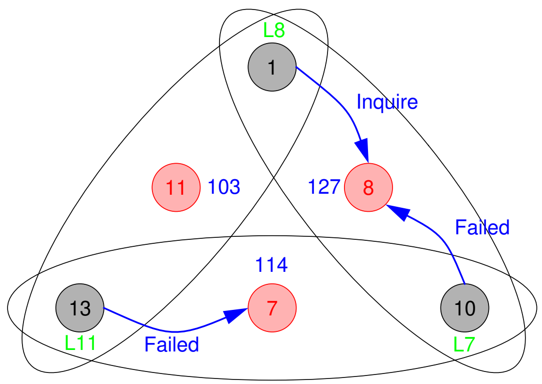
Die folgenden Skizzen zeigen, wie Deadlocks behandelt werden. Das gewählte Beispiel entspricht dem der Vorlesung, allerdings wurden alle Knoten, die nicht an der Deadlock-Situation beteiligt sind weggelassen, um die Situation zu verdeutlichen. Aus dem gleichen Grund wurden die Knoten umgeordnet:



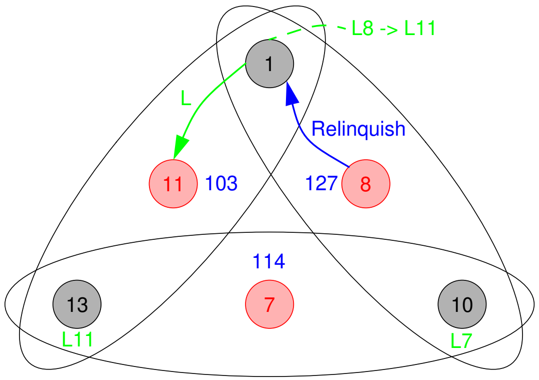
Die Knoten 11, 7 und 8 wollen in den kritischen Abschnitt und schicken dazu Requests an alle Knoten ihres Quorums (auch an die hier nicht dargestellten Knoten 2, 9 und 12 werden natürlich Requests versendet)



Die im ersten Bild zuerst eingetroffenen Requests werden von den freien Knoten direkt mit einer Locked-Nachricht beantwortet. Da nun die Knoten 1, 10 und 13 bereits ihr Lock vergeben haben, können die restlichen eintreffenden Requests nicht mehr befriedigt werden (Requests 7 an 13, 8 an 10 und 11 an1) – es entsteht eine typische Deadlock-Situation (11 wartet auf Lock von 1, 1 muß aber warten, bis 8 das Lock zurückgibt, 8 seinerseits wartet auf ein Lock von10, das aber wiederum bereits vergeben ist, ..., 13 wartet auf die Freigabe des Locks durch 11, aber 11 wartet ja ebenfalls ➔ Knoten warten "zirkulär").
Geht man davon aus, daß 11 den ältesten Zeitstempel hat (103), 7 den zweitältesten (114) und 8 den jüngsten (127), so wird diese Situation folgendermaßen aufgelöst:



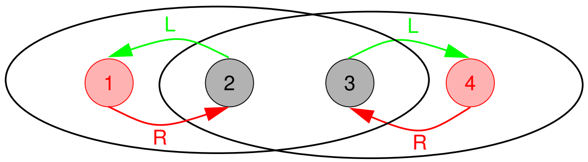
Knoten 13 beantwortet den Request von Knoten 7 mit einer Failed-Nachricht, da 13 bereits für 11 ein Lock hält und 11 einen älteren Zeitstempel als 7 besizt (d.h. der Request von 11 hat eine höhere Priorität als der von 7). Analog beantwortet 10 den Request von 8 mit Failed.
Knoten 1 bemerkt beim Eintreffen des Requests von 11, daß sein Lock für 8 ungerechtfertigt ist, da der Request von 11 älter als der von 8 ist. Somit kann potentiell ein Deadlock vorliegen und 1 fordert durch senden einer Inquire-Nachricht an 8 die Rückgabe des Locks.
Knoten 8 erhält also sowohl eine Failed- als auch eine Inquire-Nachricht, was eine Deadlock-Situation anzeigt und Knoten 8 dazu veranlaßt, das Lock an den Knoten 1 zurückzugeben:



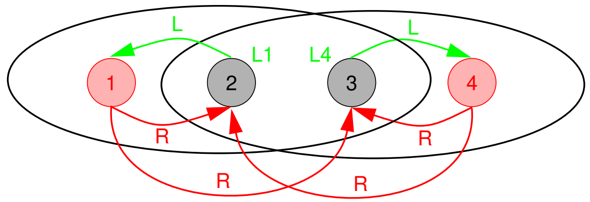
Nachdem Knoten 1 von 8 das Lock zurückerhalten hat (Relinquish), kann Knoten 1 das Lock an Knoten 11 weitergeben, so daß dieser nun alle nötigen Locks besitzt und in den kritischen Abschnitt eintreten kann.

Verläßt Knoten 11 seinen kritischen Abschnitt, sendet er Release-Nachrichten an alle Knoten seines Quorums. Diese prüfen nun ihre lokalen Anforderungsmengen und geben das Lock an die entsprechenden Knoten weiter (im obigen Beispiel 13 an 7 und 1 an 8).

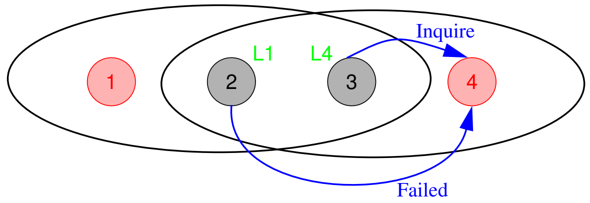
Ein anderes Szenario, das auftreten kann, wenn die Anzahl der Knoten nicht der in der Vorlesung vorgestellten Bedingung entspricht und zwei Quoren mehr als einen Knoten gemein haben, kann ebenfalls zu Deadlocks führen:



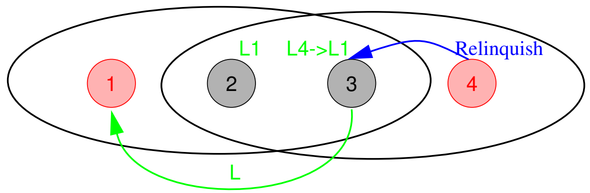
Knoten 1 und 4 wollen beide in den kritischen Abschnitt und befragen dazu die Knoten in ihren jeweiligen Quoren. Die Anfragen von Knoten 2 und 3 werden wie in der Skizze ersichtlich mit Lock-Nachrichten positiv beantwortet.



Requests werden ebenfalls von Knoten 1 an 3 und von Knoten 4 an 2 geschickt. Nachdem jedoch Knoten 2 und 3 bereits Lock-Nachrichten vergeben haben, können sie die Requests nicht befriedigen - es entsteht eine typische Deadlock-Situation. Wenn wir davon ausgehen, daß die Requests von Knoten 1 einen älteren Zeitstempel tragen als die Requests von Knoten 4, so wird diese Situation wie folgt aufgelöst:



Knoten 4 erhält von Knoten 2 eine Fail-Nachricht, da sein Request jünger als der Request von 1 ist. Außerdem erhält er von Knoten 3 eine Inquire-Nachricht, mit der das bereits vergebene Lock (Skizze 1) wieder zurückgefordert wird.



Nachdem Knoten 4 noch nicht alle notwendigen Lock-Nachrichten erhalten hatte, wird er auf die Inquire-Nachricht hin sein Lock an 3 wieder zurückgeben. Dadurch kann 3 dieses Lock an Knoten 1 weitergeben, so daß dieser in den kritischen Abschnitt eintreten kann.

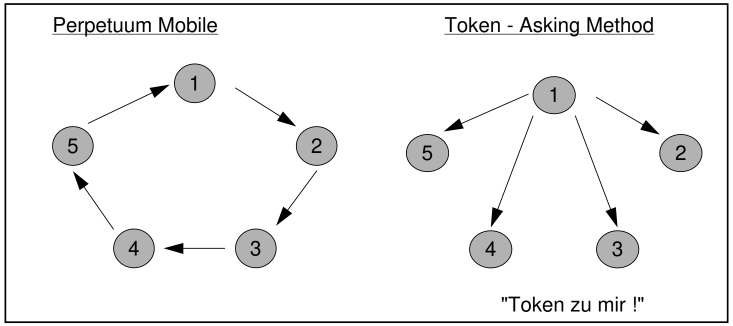
b) Was repräsentiert der Algorithmus von Sanders und worin liegt sein entscheidender Vorteil?

Der Algorithmus von Sanders ist eine Generalisierung zu den bisher behandelten erlaubnisbasierten Algorithmen zum gegenseitigen Ausschluß. Der Vorteil dieses Algorithmus liegt in seiner Anpassbarkeit an die Problemstellung: existieren Knoten, die sehr häufig in den kritischen Abschnitt treten müssen, kann die notwendige Kommunikation optimiert werden (siehe dazu auch Folie aus Vorlesung).

c) Beschreiben Sie kurz einige Algorithmen!

- Token - Perpetuum Mobile

Das Token wandert von einem Prozeß zum anderen, um ihn zu berechtigen, den kritischen Abschnitt zu betreten.
Alle Prozesse sind dabei auf einem logischen Ring angeordnet (dabei können einzelne Knoten mehrfach in dem 'logischen' Ring vorkommen -> Baumstruktur), entlang dessen das Token weitergereicht wird. Falls ein Prozeß den kritischen Abschnitt betreten will, gibt er das Token so lange nicht weiter, wie er sich in seinem kritischen Abschnitt befindet.
Ein Nachteil ist, daß viele (Token-)Nachrichten nutzlos verschickt werden, wenn der kritischen Abschnitt selten betreten werden soll.
Ein weiteres Problem stellt der Verlust und die Generierung des Tokens dar.



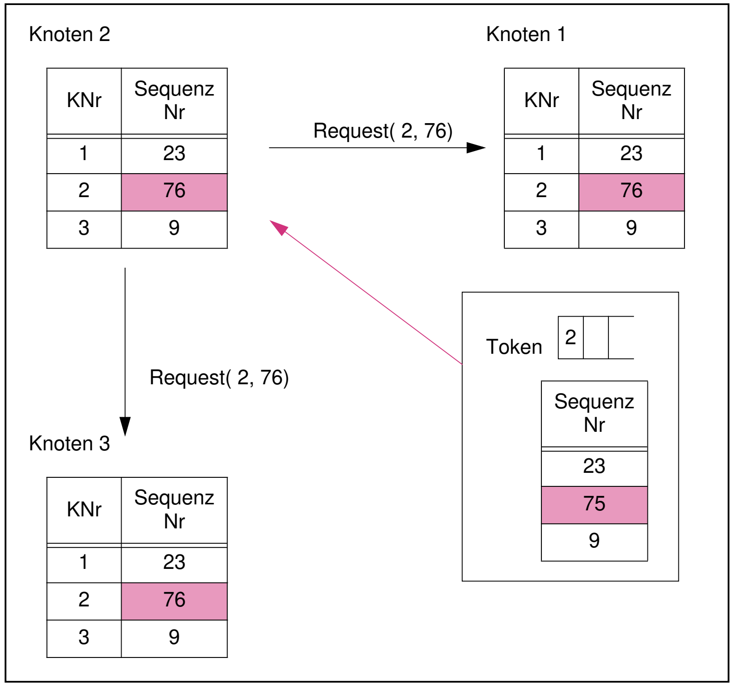
- Token - Asking Method

Das Token bewegt sich hier nicht selbständig. Ein Prozeß, der den kritischen Abschnitt betreten will, fragt nach dem Token und wartet bis er es erhält. Der Nachteil des Perpetuum Mobile (viele nutzlos zirkulierende Nachrichten) wird vermieden.
Es gibt viele Varianten, die sich darin unterscheiden, wie nach dem Token gesucht wird. Denkbar sind Broadcast (Suzuki & Kasami), ein logischer Ring, Baumstrukturen, die parallel oder sequentiell abgearbeitet werden, ein Verfolgungsalgorithmus usw.

- Suzuki / Kasami

Dieser Algorithmus arbeitet nach dem Broadcast-Prinzip. Es werden jeweils N Nachrichten benötigt: Requests werden an alle Knoten geschickt und jeder Knoten hat einen Zustandsvektor, der die Wünsche der anderen nach dem Token widerspiegelt. Auf explizite Replies kann wegen des Tokens verzichtet werden. Das Token selbst enthält einen Zustandsvektor, in dem der Zustand eines jeden Knoten zu dem Zeitpunkt festgehalten wird, zu dem das Token den kritischen Abschnitt in diesem Knoten verlassen hat:

Ablauf:
Knoten 1 besitzt das IDLE-Token, Knoten 2 fordert es an und Knoten 1 schickt es an 2.



Falls keine Anforderungen anderer Knoten vorliegen, kann ein Knoten sofort wieder in den kritischen Abschnitt, ohne andere fragen zu müssen. Der Algorithmus ist einfach und effizient.

- <u>Singhal</u>

Dieser Algorithmus stellt eine Weiterentwicklung des Algorithmus von Suzuki/Kasami dar. Dort werden Requests an <u>alle</u> Knoten verschickt. Die grundlegende Idee ist hier das Verschicken der Requests nur an solche Knoten, die das Token haben könnten (die einen Request gestellt haben).

Die Tabellen enthalten Sequenznummern und eine Information über den Zustand der anderen Knoten (*Anm: #H+#E == 1 == #Token):

Knoten i

KNr	Zustand	Sequenz Nr
1	Request	23
2	Executing*	76
3	Holding*	9
4	None	0

Token

KNr	Zustand	Sequenz Nr
1	Request	23
2	None	76
3	None	9
4	None	0

Für das weitere Verständnis des Algorithmus sind vor allem die Zustandsinformationen aller Knoten interessant. Nach der Initialisierung ergibt sich folgender Zustand:

S_1	S_2	S_3	S_4	Token
H	R	R	R	N
N	N	R	R	N
N	N	N	R	N
N	N	N	N	N

Zu beachten ist, daß die Request-Einträge ein Dreieck bilden und jeder Knoten einen Request-Hinweis auf den Knoten eingetragen hat, der gerade das Token besitzt (1. Zeile!). Die Anzahl der eingetragenen Requests ordnet die Knoten bezüglich der Zeit ihres letzten Requests. Je mehr Requests in der Spalte vorkommen, desto länger liegt die letzte eigene Ausführung des kritischen Abschnitts zurück.
Diese drei Eigenschaften bleiben während des gesamten Ablaufs des Algorithmus erhalten.

Will ein Knoten in den kritischen Abschnitt, so erhöht er seine Sequenznummer und setzt seinen Zustand auf Request. Dann sendet er eine Request-Nachricht an alle Knoten, von denen er vermutet, daß sie das Token haben könnten. Es sind alle verdächtig, bei deren Zustand er bei sich ein Request vermerkt. hat. (Anmerkung: Request-Nachrichten werden, wenn man die sortierte Matrix betrachtet, immer in die linke Richtung versendet. Dadurch kommt es zu der Ersparnis in der Nachrichtenzahl). Die Knoten, die diese Nachricht erhalten, tragen den Request im passenden Feld ihres Zustandsvektors ein. Im Beispiel wollen Knoten 3 und Knoten 2 den kritischen Abschnitt betreten:

S_1	S_2	S_3	S_4	Token
H	R	R	R	N
R	R	R	R	N
R	R	R	R	N
N	N	N	N	N

Knoten 2 sendet nach erhalt der Request-Nachricht von 3 eine Request-Nachricht an Knoten 3, da auch er in den kritischen Abschnitt treten will.

S_1	S_2	S_3	S_4	Token
N	R	R	R	N
R	R	R	R	N
R	R	E	R	R
N	N	N	N	N

Knoten 1 erhält die Nachricht von Knoten 3 vor der Nachricht von Knoten 2. Er aktualisiert die Zustandsinformation des Token und schickt es an Knoten 3, worauf dieser den kritischen Abschnitt betreten kann.
Beachten Sie, daß das Token in seiner zweiten Zeile noch veraltete Information trägt, da es von Knoten 1 sofort beim Erhalt der Request-Nachricht von Knoten 3 (also vor dem Eintreffen des Requests von Knoten 2) losgeschickt wurde.

S_1	S_2	S_3	S_4	Token
N	R	N	R	N
R	E	R	R	R
R	R	N	R	N
N	N	N	N	N

Nach Verlassen des kritschen Abschnittes aktualisiert Knoten 3 seine Vektor-Werte und die Werte des Tokens (durch die Sequenznummern kann identifiziert werden, welche Information aktueller ist). Der Zustand des Knotens wird auf None gesetzt. Da nach dem Abgleich bei Knoten 3 und im Token ein Request von Knoten 2 vermerkt ist, schickt Knoten 3 das Token an Knoten 2 weiter.

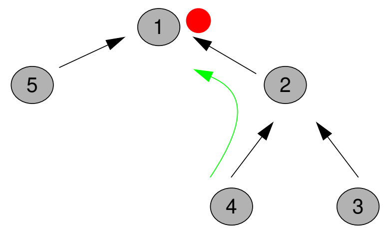
Nach dem Verlassen des kritischen Abschnitts setzt Knoten 2 seinen Zustand auf None, überprüft seine Sequenznummern mit denen des Tokens und aktualisiert beide. Da aufgrund der Sequenznummern kein weiterer Request vorliegt, behält Knoten 2 das Token und setzt seinen Zustand auf Holding

S_1	S_2	S_3	S_4	Token
N	N	N	R	N
R	H	R	R	N
R	N	N	R	N
N	N	N	N	N

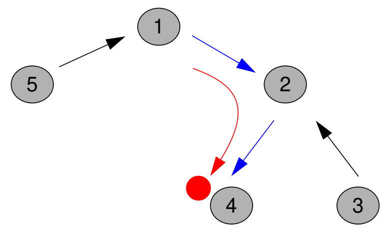
Durch eine geschickte Permutation der Knotennummern kann wieder eine Dreiecksform der Request-Zustände erzielt werden (z.B. 2->1', 3->2', 1->3', 4->4') und jeder Knoten hat einen Request-Hinweis auf den Knoten eingetragen, der gerade das Token besitzt (2. Zeile!).
Der Algorithmus zeigt seine Stärken, wenn es sogenannte Hot Spots gibt, d.h. eine Teilmenge von Knoten will den kritischen Abschnitt wesentlich öfter betreten als andere. In diesem Fall werden Request im wesentlichen nur innerhalb dieser Teilmenge ausgetauscht, z.B. die Knotenfolge 1,3,1,3,1,3,3,1,3,1,1,3,3,3,1.

- Raymond

  Die Knoten werden in Form eines spannenden Baumes angeordnet und die Pfeile zeigen in die Richtung des Tokenbesitzers.



Requests werden entlang der Kanten verschickt und das Token wandert zu dem, der es angefordert hat. Dabei werden die Kanten umgedreht. Der Knoten, der das Token besitzt stellt also immer die Wurzel des virtuellen Baumes dar.



d) Stellen Sie Kriterien auf, nach denen die Verfahren beurteilt werden können!

- Anzahl der Nachrichten
  + Einzelnachrichten (Worst Case)
  + Unter Verwendung von Broadcast (Best Case)
- Koordinierungsverzögerung (Synchronisation Delay)
  Wie viele Nachrichten müssen nach dem Verlassen des kritischen Abschnitts gesendet werden, bis der nächste hinein darf.
- Antwortzeit
- Durchsatz
- Netzbelastung, Engpässe
- Ist ein reihenfolgeerhaltender Nachrichtenmechanismus erforderlich (FIFO-Kanäle)?
- Ist eine logische Uhr notwendig?

e) Wenden Sie die von Ihnen aufgestellten Kriterien auf die Algorithmen an!

Für die folgenden Überlegungen gelten folgende Bezeichnungen:
N Anzahl der beteiligten Knoten
T Nachrichtenlaufzeit
E Verweilzeit im kritischen Abschnitt

Anzahl der Nachrichten pro Eintritt in den kritischen Abschnitt:

	Broadcast-Requests	Requests	Replies	Releases	Sonstige (worst case)	Summe Nachrichten
Zentral	1	1	1	1	-	3
Lamport	1	N-1	N-1	N-1	-	3N-3
Ricart/ Agrawala	1	N-1	N-1	-	-	2N-2
Maekawa (ideal)	-	$\sqrt{N}-1$	$\sqrt{N}-1$	$\sqrt{N}-1$	$2(\sqrt{N}-1)$	$5(\sqrt{N}-1)$
Perp. Mobile	-	-	-	1	0..∞	1..∞
Suzuki/ Kasami	1	N-1	1	-	-	N
Singhal	-	N/2-1	1	-	-	N/2
Raymond	-	log N	log N	-	-	2 log N

Wird die Anzahl der zu sendenden Nachrichten betrachtet, so ist eindeutig der zentralisierte Algorithmus im Vorteil, da er konstant 3 Nachrichten benötigt, während die anderen Algorithmen immer eine von der Anzahl der Knoten abhängige Zahl benötigen. Ausnahmen bilden die Algorithmen von Singhal und Raymond: Der Algorithmus von Singhal adaptiert sich, so daß Knoten, die längere Zeit keine Anfrage gestellt haben, nicht am Algorithmus teilnehmen. Beim Algorithmus von Raymond kann eine Optimierung durch die Wahl der Baumstruktur erreicht werden (Knoten, die häufig den kritischen Abschnitt betreten wollen, sollten durch möglichst kurze Pfade verbunden sein).

Anwortzeit und Durchsatz:

	Antwortzeit	Koordinierungs-verzögerung	Durchsatz
Zentral	2T + E	2T	$\frac{1}{2T+E}$
Lamport	2T + E	T	$\frac{1}{T+E}$
Ricart/Agrawala	2T + E	T	$\frac{1}{T+E}$
Maekawa (ideal)	2T + E	T..2T	$\frac{1}{2T+E}$
Perp. Mobile	?	$\approx \frac{NT}{2}$	$\approx \frac{1}{\frac{NT}{2}+E}$
Suzuki/Kasami	E	T oder 0	$\frac{1}{T+E}$
Singhal	E	T oder 0	$\frac{1}{T+E}$

Die Koordinierungsverzögerung bei den hier betrachteten verteilten Algorithmen ist im allgemeinen kleiner als für den zentralisierten Algorithmus. Damit ist der mittlere Durchsatz höher.

Sonstiges:

	Engpässe	Logische Uhr	FIFO-Kanäle
Zentral	1	nein	nein
Lamport	N	ja	ja
Ricart/ Agrawala	N	ja	nein
Maekawa (ideal)	?	ja	nein
Perp. Mobile	N	nein	nein
Suzuki/Kasami	N	ja	nein
Singhal	<= N/2	ja	nein
Raymond	Pfad zur Wurzel	nein	nein

Bei der Frage nach den FIFO-Kanälen bei Maekawa muß sichergestellt werden, daß INQUIRE-Nachrichten, die REPLY-Nachrichten überholt haben, so lange aufgehoben werden, bis diese eintreffen.