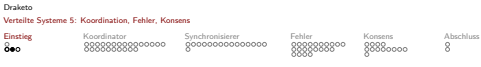


Willkommen bei Verteilte Systeme!

Von Datenbanken über Webdienste bis zu p2p und Sensornetzen.



Heute: Koordinator, Synchronisierer, Fehlertoleranz, Konsens.



Literatur

Distributed Systems - An Algorithmic Approach - Sukumar Ghosh (2015).



Ziele für Koordination

- Sie erkennen Algorithmen zur Wahl des Koordinators — leader election



Wahl eines Koordinators

- Vereinfacht viele Algorithmen
- Koordinator steuert das System
- Wenn ein Koordinator stirbt, wird ein neuer gewählt
- Wenn sich zwei Netze verbinden, wählen die Knoten einen gemeinsamen Koordinator



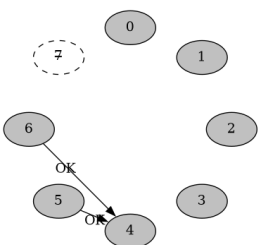
Bully-Algorithmus: Ablauf

Nachrichten: election, reply, leader

- election an alle mit höherer ID: „Kann ich Koordinator sein?“
- Warten auf reply.
 - Falls min. 1 reply: Warte auf leader.
 - Falls timeout oder keine höhere ID bekannt: leader-Nachricht an alle.
- Bei election Anfrage: Reply und weiter bei 1.
- Falls kein leader nach reply (mit timeout): Neustart.



Connected Bully - Beispiel 2



Zusammenfassung von Vorlesung 4 I

- Replikation
 - Single, Multi, Leaderless
 - Quorum Bedingung: $r + w > n$
 - kann zu Inkonsistenzen führen
 - Konsistenzmodelle definieren Garantien
 - Beispiel: Sticky Available
- CRDTs und CALM
 - Koordination vermeiden oder zumindest einschränken.
 - benötigen für Performance Garbage Collection \Rightarrow Koordination
 - Genutzt in Riak, Redis, Dynamo



Ablauf heute

- Koordinator
- Synchronisierung

--- PAUSE 14:15 ---

- Fehlertoleranz
- Konsens



Online-Versuch: Warum Koordination?

- Schalten Sie bitte Ihre Mikrophone an
- Bis 10 zählen
- Es spricht immer nur Einer oder Eine
- Wenn zwei sich unterbrechen, fangen wir neu an



Konzeptuell

Für die Algorithmen zur Wahl muss gelten:

- Alle korrekt funktionierenden Knoten eines Netzes haben den selben Koordinator.
- Der Koordinator ist Teil des Netzes.
- Der Koordinator funktioniert korrekt.

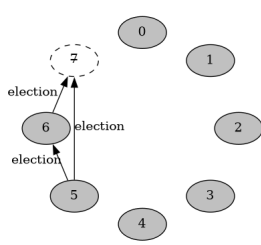


Connected Bully Algorithmus

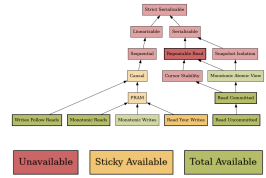
- N Prozesse $\{P_0 \dots P_{N-1}\}$.
- Wenn P_x bemerkt, dass der Koordinator nicht reagiert:
 - Sende WAHL Nachricht an alle Prozesse mit größerer ID ($\{P_{x+1} \dots P_{N-1}\}$).
 - Wenn niemand antwortet, gewinnt P_x die Wahl und wird Koordinator.
 - Wenn ein höherer Prozess antwortet, scheidet P_x aus der Wahl aus.
- Ist die Wahl beendet, werden alle Prozesse informiert.



Connected Bully - Beispiel 3



Zusammenfassung von Vorlesung 4 II



Koordinator

Verteilte, asynchrone Handlungen für ein gemeinsames Ziel



Präsenz-Versuch: Warum Koordination?

- Schließen Sie bitte die Augen
- Versuchen Sie, bis 10 zu zählen
- Es spricht immer nur Einer oder Eine
- Jede Person darf nur eine Zahl nennen
- Wenn zwei sich unterbrechen, fangen wir neu an

Wie lange brauchen wir?



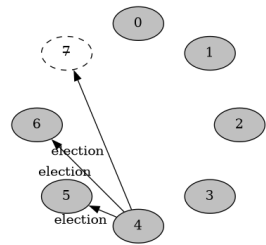
Bully-Algorithmus

- Auswahl nach ID: laufender Prozess mit höchster ID wird Koordinator
- Annahmen:
 - Vollständig verbundenes Netz, alle erreichbar und bekannt
 - Fehlerfreie Kommunikation
 - Einzigster Knoten-Defekt: Sterben
 - Es gibt einen Mechanismus zur Erkennung von Defekten
 - Es gibt eindeutig sortierte IDs.
 - Die IDs sind allen bekannt

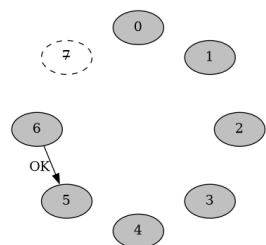
Von Garcia-Molina (1982).



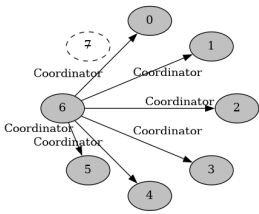
Connected Bully - Beispiel 1



Connected Bully - Beispiel 4



Connected Bully - Beispiel 5



Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Ring basierte Algorithmen

- Prozesse sind in einem Ring angeordnet
- Wahl starten: WAHL Nachricht an Nachfolger
 - Ausfallende Knoten werden übersprungen
- Verschiedene Algorithmen für lokale Entscheidung, welche ID gesendet wird
- Erreicht einen Knoten die eigene ID, sendet dieser eine COORDINATOR Nachricht um den Ring

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Maxima auf Ring-Topologie

- Bully wählt höchste ID auf verbundenem Netz → alle können alle erreichen.
- Auf Ring:
 - unidirektional : Chang-Roberts
 - bidirektional : Franklin
 - unidirektional : Peterson - in $O(N \log(N))$

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Maximum auf unidirektionalem Ring nach Chang-Roberts

- Alle starten aktiv
- schicken Token: Prozess-ID.
- verschlucken Token mit niedrigerer ID.
- Wenn sie ein höheres Token erhalten, sind sie nicht das Maximum, leiten weiter.
- Wenn sie ihr eigenes Token erhalten, sind sie das Maximum und schicken ein Leader token.

Worst case: $O(N(N+1)/2)$ Nachrichten.

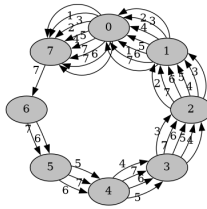
Was ist der Worst-Case?

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Chang-Roberts: Worst Case



Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Online-Versuch: Unidirektionaler Chat

- Ich gebe den Bildschirm frei
- Meine Matrix ist die Reihenfolge
- Erzeugt euch eine zufällige ID auf <https://www.random.org/integers/?num=1&min=1&max=10000&col=1&base=10&format=html&rnd=new>
- Schreibt im Chat an die nächste Person in der Reihe
- Wir nutzen Chang-Roberts, um die Person mit der höchsten ID zu finden

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Maximum auf Ring nach Franklin

- Wie Chang-Roberts, aber in beide Richtungen.
- In Runden¹
- Jede Runde wird mindestens die Hälfte der Prozesse inaktiv.

$O(\log(N))$ Runden → Worst-Case: $O(N \log(N))$ Nachrichten.

Was ist der Worst-Case?

¹Eine Runde: Alle Knoten gehen einen Schritt weiter. Synchronisiert.

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Maximum auf Ring nach Peterson

- Schicke jede Runde mein Alias und das meines Vorgängers
- Erhalte jede Runde das Alias meines Vorgängers und des Vor-Vorgängers
- Wenn das Alias meines Vorgängers größer ist als meins und als das des Vor-Vorgängers, nimm das des Vorgängers an und bleibe aktiv.
- Ansonsten werde inaktiv (leite nur noch weiter)
- Zwei Vergleiche pro Runde → wie Franklin!

Worst-Case: $O(N \log(N))$ Nachrichten, Koordinator hat höchstes Alias, aber nicht höchste Prozess-ID — wurde weitergeleitet!

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Maximum auf beliebiger Topologie

- Fluten, aber sende nur die höchste erhaltene weiter
- Anzahl Runden aus Netzwerk-Durchmesser (D) → muss bekannt sein!

Anzahl der Nachrichten: $O(\Delta D)$

Δ = maximale Zahl Nachbarn (max degree)/

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Koordinator-Wahl in anonymen Netzen

- Braucht Symmetriebruch, z.B. Zufallszahlen
- Beispiel:
 - Wirf eine Münze.
 - Bei Zahl benachrichtige alle aktiven Prozesse.
 - Bei Kopf werde passiv. Wenn du keine Nachricht erhältst, werde wieder aktiv und wiederhole.
 - Wenn du aktiv bist und keine Benachrichtigungen erhältst, bist du Koordinator.

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Ohne volles Vertrauen an Alle

- Komplexer
- Commit-Reveal-Protokolle
- Konkrete Möglichkeit: Mental Poker

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Zusammenfassung Koordinator

- Die Wahl eines Koordinators erleichtert den Algorithmus-Entwurf.
- Je nach Topologie unterschiedliche Algorithmen.
- Petersons Algorithmus erreicht auf einem unidirektionalen Ring die Skalierung des Bidirektional, tauscht dafür allerdings IDs aus.
- Netze ohne IDs brauchen Symmetriebrüche.

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Ziele für Synchronisierer

- Sie kennen Methoden zur Synchronisierung

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Synchronisierer

- Teilen die Berechnung in Diskrete Schritte (ticks).
- Ermöglichen synchrone Algorithmen in asynchronen, verteilten Systemen.
- Nachrichten-Overhead oft durch günstigere Algorithmen ausgeglichen.

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Präsenz-Versuch: Zählen mit Koordination

- Schließen Sie bitte die Augen
- Koordinator-Wahl:
 - Ich nenne den Namen der startenden Person
 - Wer startet, nennt Namen Anderer früher im Alphabet
 - Wenn Sie ihren Namen hören, antworten Sie „hier“
 - Wenn niemand früher im Alphabet „hier“ sagt, sagen Sie „Koordinator“
- Zählen:
 - Als Koordinator rufen Sie Leute mit Namen auf, die hochzählen
 - Wir versuchen, bis 10 zu zählen
 - Jede Person darf nur eine Zahl nennen
 - Wenn zwei sich unterbrechen, ist die nächste Zahl wieder 1

Wie lange brauchen wir mit Koordinatorwahl?

Draketo
Verteilte Systeme 5: Koordination, Fehler, Konsens

Einstieg Ring

Koordinator Synchronisierer Fehler Konsens Abschluss

Online-Versuch: Smiley-Teppich im Chat

Ziele:

- wir schreiben im jitsi-chat erst alle :-), dann :-), dann :-D
- möglichst schnell
- alle schreiben, bevor der nächste Smiley kommt
- kein Überlappen

Zeit läuft ab ...

Aktionen pro Tick

Jeder Prozess kann:

- Berechnungen ausführen
- Nachrichten schicken

Asynchronous bounded delay (ABD)-Synchronisierer

- Braucht Uhren mit ausreichend niedrigem Drift
- Maximalverzögerung von Nachrichten: δ

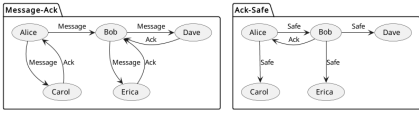
Ablauf:

- Stelle C auf 0 + sende start(C=0) an Nachbarn
- Starte C=1 erst bei $2 \cdot \delta$.



Awerbuch (α) Synchronisierer

- Funktioniert ohne Uhren und Maximalverzögerung.
- Methode: Nachrichten \rightarrow Acks \rightarrow Safe.
- Verschiedene Zeit- und Nachrichtenkomplexität



α -Synchronisierer

- Sende Nachrichten $\langle m, i \rangle$ für Tick i
- Sende $\langle \text{ack}, i \rangle$ für jede empfangene Nachricht und warte auf $\langle \text{ack}, i \rangle$ für jede deiner Nachrichten
- Sende $\langle \text{safe}, i \rangle$ für jeden Nachbar.
- Warte auf $\langle \text{safe}, i \rangle$ von jedem Nachbar.
- Starte Tick $i+1$

Nachrichtenkomplexität: $M(\alpha) = O(|E|)$.
 Zeitkomplexität: $T(\alpha) = 3 \rightarrow m, \text{ack}, \text{safe}$.
 $|E|$: Die Anzahl der Kanten (edges).



Online-Versuch: Synchronisierte Smileys

- Vollverbundenes Netz (unser Chat)
 - Ack: Gleichen Smiley eintragen. Alle Acks empfangen, wenn alle Smileys gepostet sind.
 - Safe: Hand heben
- Ablauf: $:-) \rightarrow :-) \rightarrow :-D$
- Trage den ersten oder den selben Smiley ein
 - Warte, bis alle anderen den selben Smiley eingetragen haben
 - Hebe die Hand
 - Wenn alle die Hand oben haben, senke sie wieder \Rightarrow nächster Smiley



Online-Versuch: Ist das genau α ?

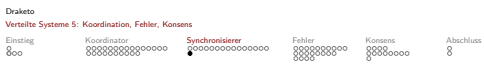
- Ack für Nachricht — Smiley sehen?
- Safe sehen — alle Smileys sehen?
- Was ist „ich sehe alle Smileys“?

Brauchen wir hier einen vollen α -Synchronisierer?



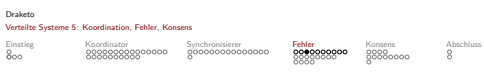
β -Kosten

Nachrichtenkomplexität: $M(\beta) = O(n)$ – statt $|E|$ für α .
 Zeitkomplexität: $T(\beta) = \Omega(\log(n))$, worst case: $(n-1)$.



Zusammenfassung Koordination

- Die Wahl eines **Koordinators** erleichtert den Algorithmus-Entwurf.
- **Synchronisierer** ermöglichen die Nutzung der einfacheren **synchronen Algorithmen** in asynchronen Systemen.



Arten von Fehlern

- Crash** Endet für immer
- Auslassung** Verliert Nachrichten (omission)
- Vorübergehend** Verändert den globalen Zustand zufällig (transient)
- Byzantinisch** Jede vorstellbare Art fehlerhaften Verhaltens
- Software** Verschiedene Gründe (nächste Folie)
- Zeitlich** Deadline verpasst
- Sicherheit** Viren, Trojaner, Würmer, ...
- Heisenbugs** Nicht verlässlich reproduzierbar



Fehlertolerante Systeme

- Maskierend** *Sicherheit + Lebendigkeit* \rightarrow Flugzeugturbine (kann weiterfliegen)
- nicht-maskierend** nur Lebendigkeit, Sicherheit zeitweise nicht \rightarrow GC pause
- Fail-safe** nur Sicherheit \rightarrow Raketenstart abbrechen
- Graceful degradation** Noch akzeptabel \rightarrow nächste Folie



Asynchrone Komplexität mit Synchronisierer

$$M_A = M_S + T_S \cdot M(A) \quad (1)$$

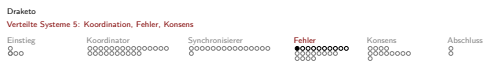
$$T_A = T_S \cdot T(\alpha) \quad (2)$$

Zusätzliche Nachrichten pro Zeitschritt.
 Multiplikator der Zeit.



γ -Synchronisierer

- Netz in Cluster aufteilen
- Jeden Cluster hierarchisch via β -Synchronisierer
- Zwischen Clustern α -Synchronisierer



Fehler-Tolerante Systeme

Fehler sind unvermeidbar. Problem: Häufigkeit von Fehlern.

Ziele:

- Sie kennen verschiedene Klassen von Knoten-Defekten
- Sie kennen Eigenschaften von Systemen zur Erkennung von Knotenverlusten
- Sie wissen um selbststabilisierende Algorithmen



Software-Fehler

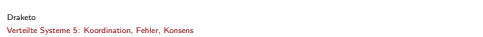
- Coding/menschlich** 23. September 1999 rechnete die NASA die Höhe über dem Mars in Fuß und Metern
- Design** Vertauschte Prioritäten im Mars pathfinder real-time kernel — Kommunikation verhungerte.
- Memleaks** Verbraucht Ressourcen
- Spezifikationsfehler** Fehler in Annahmen. Beispiel: Annahme: Ich kann mein Objekt über JSON serialisieren. Realität: JSON-keys sind immer Strings.

```
# python
a = {1: 2}
json.loads(json.dumps(a))
# {'1': 2}
```



Beispiele für graceful degradation

- Taxi: ?
- Paketweiterleitung: ?
- Kaffeeautomat: ?
- Dateisystem: ?



β -Synchronisierer

- Wähle Koordinator.
- Spanning tree, Koordinator ist Wurzel.
- Koordinator started Tick i mit $\langle \text{next}, i \rangle$ den Baum entlang.
- Knoten senden $\langle m, i \rangle$ an Zielprozesse.
- Knoten senden und empfangen $\langle \text{ack}, i \rangle$ für Nachrichten.
- Knoten senden $\langle \text{safe}, i \rangle$ an Eltern.
- Koordinator wartet auf $\langle \text{safe}, i \rangle$, startet dann Tick $i+1$.



Zusammenfassung Synchronisierer

Synchronisierer ermöglichen die Nutzung der einfacheren synchronen Algorithmen in asynchronen Systemen.

$$\text{Zeit: } T_A = T(x) \cdot T_S \quad (3)$$

$$T(\alpha) = 3; \text{ message, ack, safe} \quad (4)$$

$$T(\beta) = O(\text{height}); \text{ path to root} \quad (5)$$

$$\text{Nachrichten: } M_A = M_S + T_S \cdot M(x) \quad (6)$$

$$M(\alpha) = O(|E|) \quad (7)$$

$$M(\beta) = O(n) \quad (8)$$

$$(9)$$



Fehler: Definition

Fehler Nicht-erwartetes Verhalten.
Fehlertoleranz Maskiert Fehler oder stellt das erwartete Verhalten wieder her.



Beispiel-Fehler

```
define : faulty-system-1
define x #t
while-any
x : send 'correct
#t : send 'faulty
```

Scheduler: Schwach fair \rightarrow Fehler wird garantiert sichtbar.



Progressive improvement

- Aktuell für Webseiten verwendet
- Umgekehrte graceful degradation
- Erst Basisdienst definieren und implementieren
- Für bestimmte Plattformen bessere Qualität



Fehlertoleranz (Minimal)

- Crash Redundanz
- Auslassung Bestätigungen → Sequenznummern (TCP)
- Andere Fail-safe + Crash

Zusammenfassung Fehler

- Fehlerhäufigkeit minimieren
- Fehlerarten: Crash, Auslassung, Vorübergehend, Byzantinisch, Software, Zeitlich, Sicherheit, Heisenbugs
- Toleranz: Maskierend, nicht-maskierend, Fail-safe, Graceful degradation

Erkennung von Knotenverlusten

Klassifizierung von Erkennungssystemen zur Analyse.

- Vollständigkeit** Welche Prozesse werden sicher gefunden?
- Korrektheit** Gibt es Falschmeldungen? Von wie vielen?



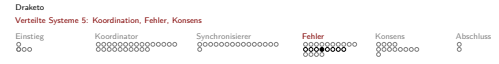
Starke Erkennung

- Vollständigkeit** Jeder verlorene Prozess wird von allen erkannt
- Korrektheit** Kein aktiver Prozess wird je verdächtigt



Schwache Erkennung

- Vollständigkeit** Jeder verlorene Prozess wird von mindestens einem erkannt und bleibt danach verdächtig
- Korrektheit** Mindestens ein aktiver Prozess wird nie verdächtig
- Aus schwacher Vollständigkeit lässt sich starke Vollständigkeit rekonstruieren.



Eventually correct

- Schwächste Form: Irgendwann gibt es mindestens einen aktiven Prozess, der nicht verdächtig wird, fehlerbehaftet zu sein.
- Aktiver Prozess heißt: Korrekt funktionierender Prozess.



Implementierung

- Üblicherweise Timeouts
- z.B. Heartbeat + Ack



Wieso das ganze?

Klassifizierung der Erkennung, um Algorithmen beweisen zu können.



Zusammenfassung Fehlererkennung

- Vollständigkeit** Wer weiß was?
- Korrektheit** Falschmeldungen?
- Implementierung** Timeouts

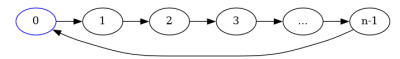
PAUSE

Selbststabilisierung

Rückführung auf gültigen Zustand als Teil des Algorithmus.

- Zeitweise Fehler: Stromschlag frisst Token
- Topologie-Änderungen: „Churn“
- Umgebungsänderungen: Morgens gültig, Abends nicht, dazwischen?

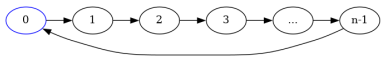
Beispiel: Tokenwiederherstellung



- Sie können durch Zählen ein fehlendes Token erkennen.
- Können Sie Tokenfehler unproblematisch machen?



Beispiel: Tokenwiederherstellung (Algorithmus)



```

define : ring i
cond
  {i = 0}
  while-any
    {(ref s 0) = (ref s n-1)}
    list-set! s 0 : +1modk (ref s 0)
  else
  while-any
    : not {(ref s i) = (ref s {i - 1})}
    list-set! s i : ref s {i - 1}
  
```

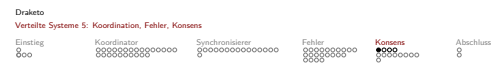
```

define N 10
define k 11 ; k > N/
define s : make-list N
define n-1 {N - 1}
define : +1modk x
  modulo {x + 1} k
define ref list-ref
Dijkstra (1974)
  
```



Zusammenfassung Fehler

- Wichtigste Fehlerarten: Crash, Auslassung, Byzantinisch.
- Wichtigste Fehlertoleranz: Maskierend, nicht-maskierend.
- Crash-Erkennung: Klassifiziert nach Vollständigkeit und Korrektheit
- Selbststabilisierung: Korrektur von Fehlern Teil des Algorithmus



Konsens

Eine gemeinsame Entscheidung treffen.

- Ziele:
 - Sie verstehen die Herausforderungen der verteiltem Konsensfindung
 - Sie können zwei Beispiele für verteilten Konsens nennen

Bedingungen an einen Algorithmus

- (Prozesse: P, nicht-schadhafte: P*):
 - Endet** Alle P* müssen irgendwann entscheiden (termination)
 - Einigkeit** Alle P* entscheiden gleich (agreement)
 - Gültigkeit** Wenn alle P* mit dem gleichen Anfangswert v beginnen, muss die Entscheidung v sein (validity)
 - Endgültigkeit** Nachdem die Entscheidung getroffen ist, bleibt sie für immer

Konsens in asynchronen Systemen

- Trivial in fehlerfreien Systemen:
 - Verteile alle Einzelentscheidungen
 - Wende gleiche Entscheidungsfunktion an
- Mit Fehlern wird es spannend.

Garantierte Entscheidung mit Crash unmöglich

- Asynchrones verteiltes System → Beliebige Verzögerungen.
- Zustände mit Zünglein an der Waage (Entscheider).
- Was, wenn das Zünglein zögert?
- Es gibt immer einen Entscheider oder eine Entscheiderin, auch wenn oft unbekannt.
- In absolut asynchronen Systemen ist ein Crash nicht von Verzögerung unterscheidbar.

Die Byzantinischen Generäle

Konsens in einem synchronen verteilten System mit byzantinischen Fehlern.

- Angriff oder Rückzug?
- Es kann Verräter geben.

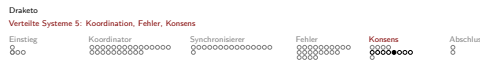
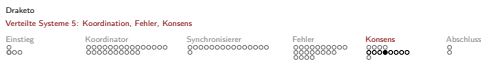
Lösung ohne Verräter

- Entscheidungen verteilen
- Identische Entscheidungsfunktion anwenden

Anforderungen an einen Algorithmus mit Verrättern

Kommandant und Lieutenant:

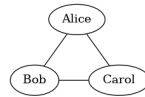
- 1 Jeder loyale Lieutenant erhält den gleichen Befehl
- 2 Wenn der Kommandant loyal ist, erhält jeder loyale Lieutenant den Befehl des Kommandanten



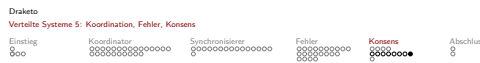
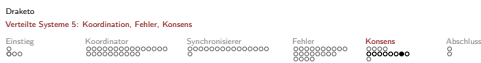
Lösung mit Verrättern und Wortnachrichten

- Nachrichten werden nicht korumpiert
- Nachrichten können verloren gehen, aber ihr Fehlen kann erkannt werden
- Bei Erhalt ist die Identität des Senders bekannt

Bei 3 Generälen gibt es keine Lösung mit mündlichen Nachrichten



Drei Fälle



Verallgemeinerung

Für 3 Generäle von denen einer ein Verräter ist gibt es mit verbalen Nachrichten keine Lösung.

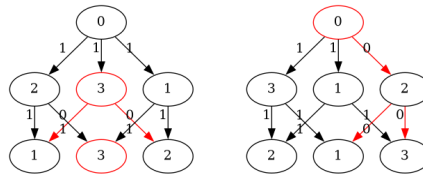
Generäle können in gleichgroße Gruppen zusammengefasst werden, mit den Verrättern zusammen in einer Gruppe.

⇒ Bei 3m Generälen und mindestens m Verrättern gibt es keine Lösung.

⇒ $N \geq 3m + 1$

Eine Lösung

- Ziel der Verräter: Eine Entscheidung verhindern.
- Lösung: Weiterleiten + Mehrheitswahl

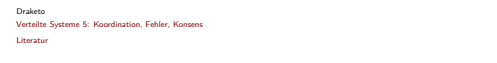
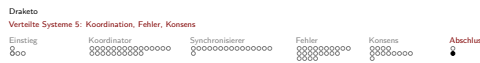
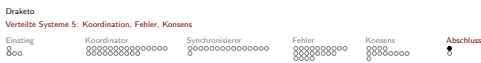


Zusammenfassung Konsens

- Byzantinische Generäle sind ein klassisches Beispiel.
- Mündlichen Nachrichten: Weniger als 1/3tel Verräter.
- Signaturen: Mindestens zwei loyale.

■ Störtoleranz über Fehlererkennung prüfbar.

Konkrete Umsetzung: Raft.



Zusammenfassung I

- Koordinator vereinfacht Algorithmen
- Synchronisierer ermöglichen synchrone Algorithmen in asynchronen Systemen
- Fehler: Crash, Auslassung, Byzantinisch
- Toleranz: Maskierend?
- Erkennung: Vollständigkeit, Korrektheit
- Selbststabilisierung
- Konsens: Byzantinische Generäle

Danke!

☺

Verweise I

Ghosh, S. (2015). *Distributed Systems - An Algorithmic Approach*. Computer & Information Science. Chapman & Hall/CRC, 2 edition, ISBN: 978-1466552975.

