Verteilte Systeme

Diffusionsalgorithmen

Diffusionsalgorithmen

Definition:

Ein verteilter Diffusionsalgorithmus auf einem Beliebigen Graphen startet in einem Zustand, in dem alle Knoten des Graphen “idle” (quiescent) sind.

Eine Nachricht (Sonde, Auftrag) weckt einen Knoten auf, der Zustandsautomat führt lokale Aktionen aus und Sendet ggf. Nachrichten um Teilaufträge an Nachbarknoten zu verteilen.

Knoten werden durch Nachrichten aufgeweckt, und sind nach Erfüllung der Teilaufträge wieder “idle”.

Irgendwann sind sind alle Teilaufträge erfüllt, der Algorithmus ist fertig.

Verteilte Terminierungserkennung

Problem:

Ein verteilter Diffusionsalgorithmus (z.B. asynchrone Breitensuche) soll ausgeführt werden.

Dabei ist es nicht Notwendigerweise der Fall, dass Knoten das Ergebnis Ihrer Teilaufgabe kommunizieren.

Wie soll die Terminierung des Algorithmus (es gibt keine Knoten, die nicht idle sind) erkannt werden?

Dijkstra Schloten Algorithmus

Idee:

Wir nutzen den Diffusionsalgorithmus als Träger, um einen MST der aktiven Knoten zu verwalten.

Knoten Warten auf Bestätigung ihrer Teilaufträge.

Sobald die Wurzel idle wird, wissen wir der Algorithmus hat terminiert.

Dijkstra Schloten Algorithmus

```
monitor proc:

  start():
    status := source
    await(waiting == 0)
    status := idle
    // done

  send(m):
    waiting++
    super.send(m)

  recv(m):
     if status == idle
        parent := src
        status := non-src
     else
        send_src(ack)

  recv(ack):
     waiting--
     if waiting == 0
        send_parent(ack)
        status := idel
```

Verteilte Verklemmungserkennung

Verklemmungen können bei Verdacht oder beim Entstehen erkannt werden.

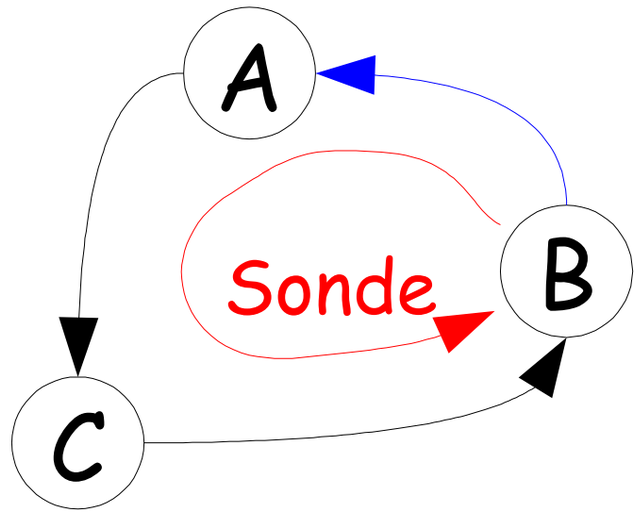
Idee:

Wir wollen Verklemmungen beim Entstehen erkennen.

Jeder Knoten führt eine Liste der Knoten, auf die er wartet.

Bevor der Prozess blockiert,

sendet er eine Sonde entlang des Wartegraphen aus (ohne Echo)



Kommt die Sonde beim Absender an, so kann eine Verklemmung vorliegen.

Verteilte Systeme

Globaler Zustand

Globaler Zustand

Idee:

Wir wollen konsistent den Zustand eines Verteilten Systems erfassen.

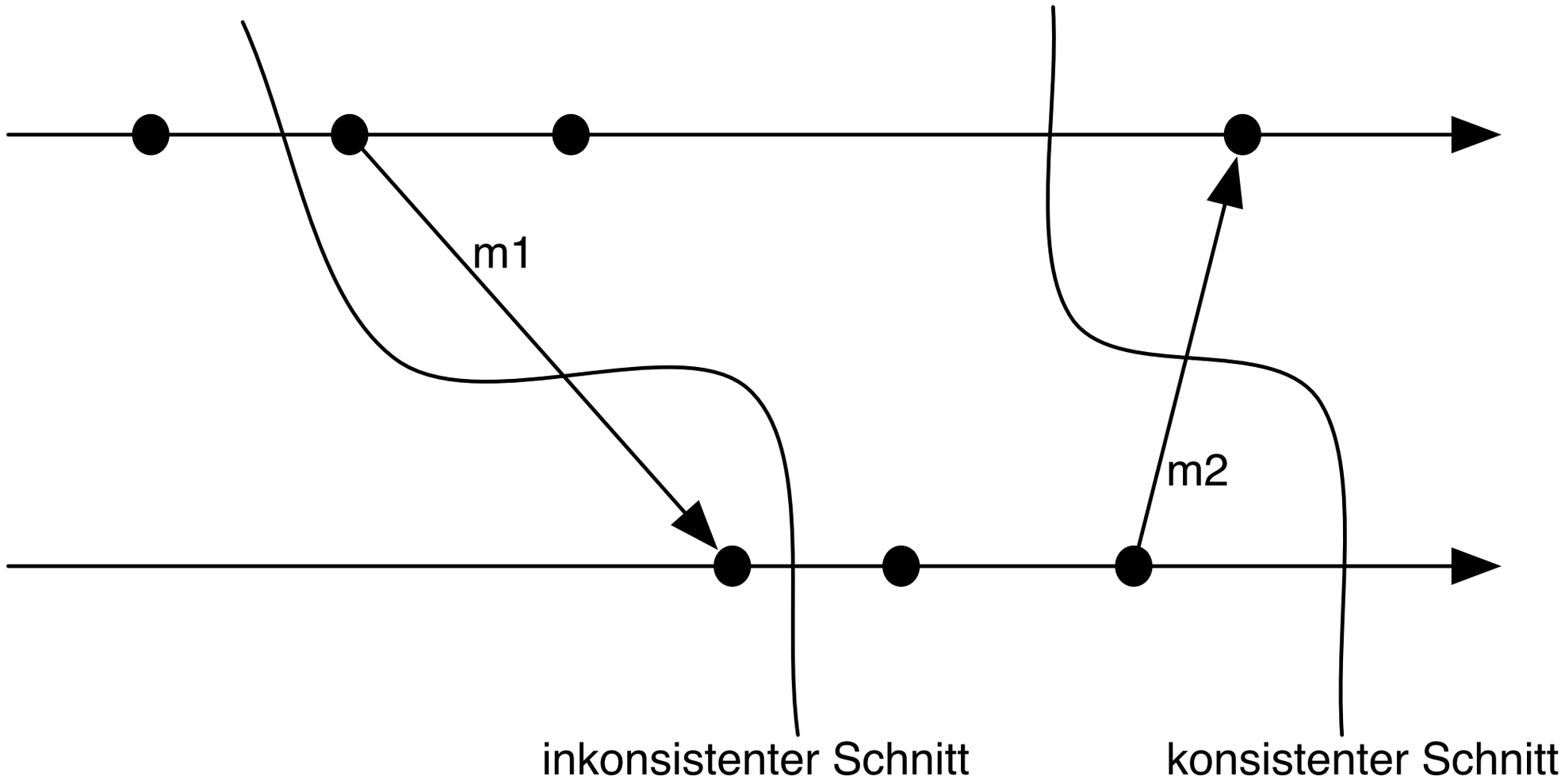
Anwendungen:

Verteiltes Debugging, Terminierungserkennung, Deadlockerkennung, …

Problem:

Was ist ein konsistenter Zustand?

Konsistente Schnitte



Konsistenter globaler Zustand

- Ein konsistenter Schnitte C sind Mengen von Ereignissen, $e_1, e_2, \ldots e_i$ für die gilt: $\forall\, e_j \rightarrow e_i : e_i \in C \Longrightarrow e_j \in C$
- Ein Konsistenter globaler Zustand korrespondiert zu einem konsistenten Schnitt und umfasst den Zustand aller Prozesse.

Algorithmus von Chandy und Lamport

Voraussetzung: Ein beliebiges Netzwerk.

Idee:

Um einen Snapshot zu erstellen fluten wir Marker durch das Netz.

Wenn ein Marker eintrifft für einen neuen Snapshot eintrifft, dann wird der Zustand des Prozesses gesichert und auf allen Kanälen wird der Marker ausgesandt.

Problem:

Wie gehen wir mit Nachrichten um, die während eines Snapshots gesendet wurden?

Algorithmus von Chandy und Lamport

Problem:

Wie gehen wir mit Nachrichten um, die während eines Snapshots gesendet wurden?

Idee:

FIFO-Eingenschaft der Kanäle nutzen:

Alle Nachrichten, die nach dem ersten Marker, aber vor dem Marker auf dem Kanal empfangen wurden werden als Teil des Snapshots gespeichert.

Wenn über alle Kanäle ein Marker empfangen wurde, ist der Snapshot fertig.

Algorithmus von Chandy und Lamport

Kommunikationskomplexität: O(|E|)

Zeitkomplexität: O(|V|*(s+d))

Konsistenter globaler Zustand / Prädikate

- Wir können den Zustand eines VS durch den Zustand der Prozesse und der Nachrichten "in transit" beschreiben.

- Ein Prädikat ist eine Aussage auf dem Zustand
  z.B. es gibt einen Prozess, der idle ist

- Prädikate können dann verteilt oder zentral von einem Koordinator geprüft werden.

- Wir nennen ein Prädikat ist stabil, wenn es keinen erreichbaren Folgezustand des VS gibt, in dem dieses Prädikat nicht mehr gilt (z.B. Terminierung, Deadlock)

Verteiltes Debugging

- Ein ausgezeichneter Knoten löst an einem Brakepoint einen Snapshot aus
- Der Zustand des Systems beim Erreichen des Brakepoints kann anschließend analysiert werden.

Verteilte Verklemmungserkennung

- Eine Verklemmung ist ein stabiler zustand, der bei Bedarf erkannt werden kann.
- Auf einem Snapshot kann der Wartegraph aufgestellt werden und z.B. mit Breitensuche auf Kreise untersucht werden.

Verteilte Garbage Collection

- Garbage Collection ermöglicht es dynamischen Speicher zu verwalten, ohne dass nicht mehr benötigte Objekte explizit freigegeben werden müssen.

- Garbage-Collection mit Referenzzählern sind im verteilten Fall aufwändig (explizite Benachrichtigung des Hosts für jede Erstellung / Aufgabe einer Referenz) und Fehleranfällig (Ausfall von Stationen, Paketverlust)

- Auf einem Snapshot kann sicher überprüft werden, ob ein Objekt noch Referenziert ist
  falls nein => stabiler Zustand

Verteilte Systeme

CAP-Theorem

CAP-Theorem

Es ist unmöglich ein Verteiltes System zu bauen, dass die folgenden Eingenschaften erfüllen:

- Konsistenz
- Verfügbarkeit
- Partitionstoleranz

Behauptung: Eric Brewer 2000
Beweis: Seth Gilbert & Nancy Lynch 2002

CAP-Theorem

Für Konsistenz fordern wir:

- Alle Zugriffe auf ein Objekt sind linearisierbar
  d.h. Es gibt eine sequentielle Ausführung auf einem zentralen System, die den gleichen Zustand erzeugt

CAP-Theorem

Für Verfügbarkeit fordern wir:

- Jede Anfrage an einen Verfügbaren Knoten muss mit eine gültige Antwort erhalten.

CAP-Theorem

Für Partitionstoleranz fordern wir:

- Falls Teile des VS keine Nachrichten mehr austauschen können, funktionieren die Teilsysteme trotzdem weiter.

CAP-Theorem

Beweisidee:

Beweis durch Widerspruch - unter der Annehme, das wir einen Algorithmus haben, der Konsistenz, Verfügbarkeit und Partitionstoleranz gewährleistet, erzeugen wir einen Zustand, der inkonsistent ist.

Ein System, dass das Lesen und Schreiben von Werten erlaubt, ist in zwei Partitionen (G_1, G_2) geteilt, die nicht miteinander kommunizieren können.

Ein Schreibzugriff auf G_1 muss erfolgreich sein (Verfügbarkeit).

Danach wird von G_2 gelesen, da G_2 nicht von der Schreiboperation weiss wird G_2 den alten Wert zurückgeben - Widerspruch zur Konsistenz.

Übung 5 zum 14. Juni 2011

Wir werden über die nächsten Übungszettel eine kleine verteilte Peer-to-Peer Wirtschaftssimulation schreiben.

Implementieren sie ein kleines Kommandozeilen-programm, das in der Lage ist Befehle entgegen-zunehmen und implementieren sie zwei Befehle:

connect <hostname> <port>
  stellt per UDP-Channel eine Verbindung zu einem Peer her

peers
  Zeigt alle Peers in der Zusammenhangskompoente an