Synchronisierung

Algorithmen für verteilte Systeme

Sebastian Forster

Universität Salzburg

Synchron vs. asynchron

CONGEST Modell:

- Netzwerk mit n Knoten und m Kanten
- Kommunikation mit Nachbarn in synchronen Runden
- Bandbreite (= maximale Nachrichtengröße) $O(\log n)$
- Zahlreiche Algorithmen für klassische Graphprobleme

Synchron vs. asynchron

CONGEST Modell:

- Netzwerk mit n Knoten und m Kanten
- Kommunikation mit Nachbarn in synchronen Runden
- Bandbreite (= maximale Nachrichtengröße) $O(\log n)$
- Zahlreiche Algorithmen für klassische Graphprobleme

Aber:

- Annahme synchroner Runden ist idealisiert
- In realen Anwendungen können unterschiedlich lange Delays Asynchronität hervorrufen

Synchron vs. asynchron

CONGEST Modell:

- Netzwerk mit n Knoten und m Kanten
- Kommunikation mit Nachbarn in synchronen Runden
- Bandbreite (= maximale Nachrichtengröße) $O(\log n)$
- Zahlreiche Algorithmen für klassische Graphprobleme

Aber:

- Annahme synchroner Runden ist idealisiert
- In realen Anwendungen können unterschiedlich lange Delays Asynchronität hervorrufen

Ziel

Simuliere synchrone Algorithmen in asynchronen Netzwerken mit moderaten Overheads in der Zeit- und Nachrichtenkomplexität

Asynchrones CONGEST Modell

- Event-basiert statt diskrete Zeitschritte

Asynchrones CONGEST Modell

- Event-basiert statt diskrete Zeitschritte
- Zwei mögliche Events: Initialisierung oder Empfang einer Nachricht

Asynchrones CONGEST Modell

- Event-basiert statt diskrete Zeitschritte
- Zwei mögliche Events: Initialisierung oder Empfang einer Nachricht
- Jede Nachrichtenübermittlung benötigt endliche Zeit

Asynchrones CONGEST Modell

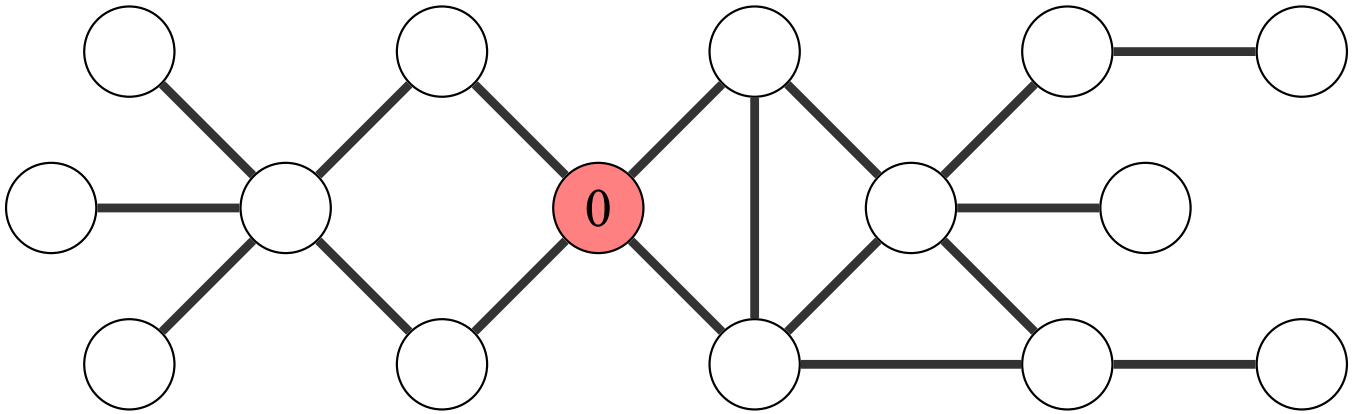
- Event-basiert statt diskrete Zeitschritte
- Zwei mögliche Events: Initialisierung oder Empfang einer Nachricht
- Jede Nachrichtenübermittlung benötigt endliche Zeit
- Für Laufzeitanalyse: Delay von höchstens einer Zeiteinheit

Asynchrones CONGEST Modell

- Event-basiert statt diskrete Zeitschritte
- Zwei mögliche Events: Initialisierung oder Empfang einer Nachricht
- Jede Nachrichtenübermittlung benötigt endliche Zeit
- Für Laufzeitanalyse: Delay von höchstens einer Zeiteinheit
- Korrektheit muss für beliebige Delays gelten

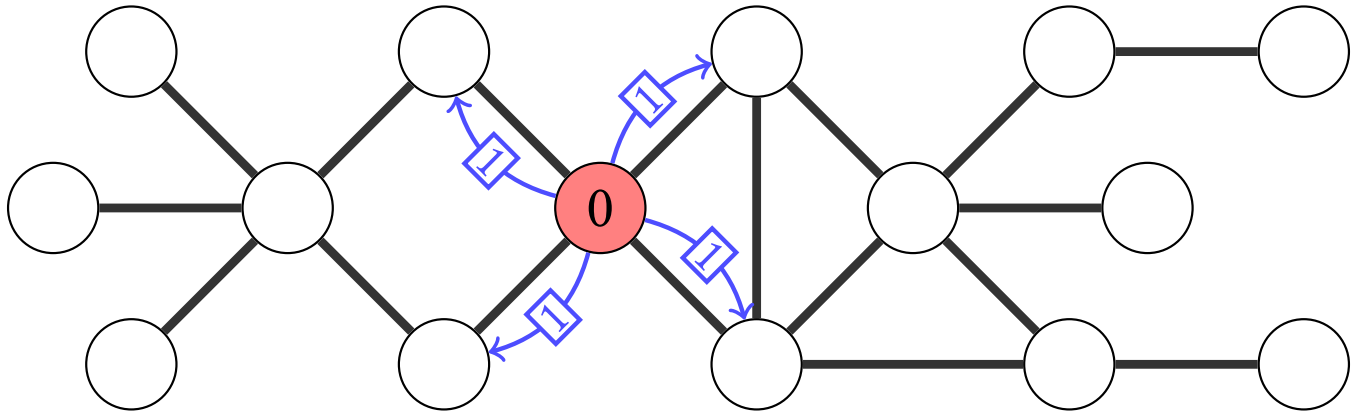
Beispiel: Breitensuche

Naive Ausführung synchroner Algorithmen ist in der Regel nicht erfolgreich (Inkonsistenzen)



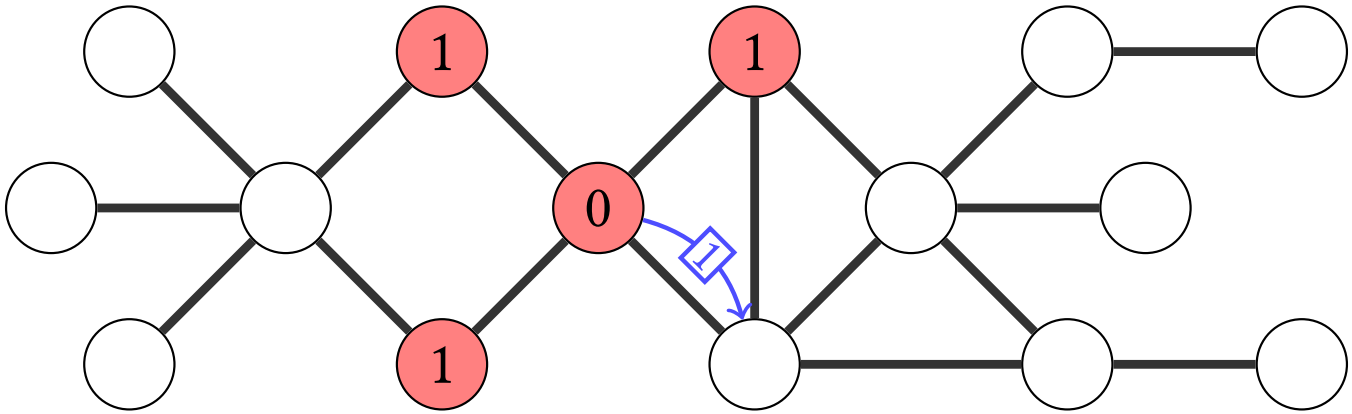
Beispiel: Breitensuche

Naive Ausführung synchroner Algorithmen ist in der Regel nicht erfolgreich (Inkonsistenzen)



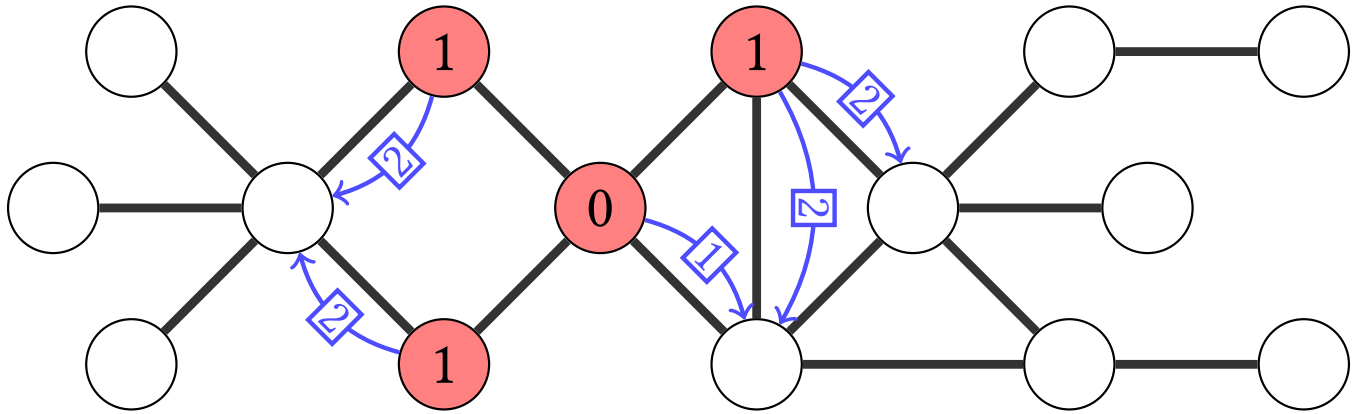
Beispiel: Breitensuche

Naive Ausführung synchroner Algorithmen ist in der Regel nicht erfolgreich (Inkonsistenzen)



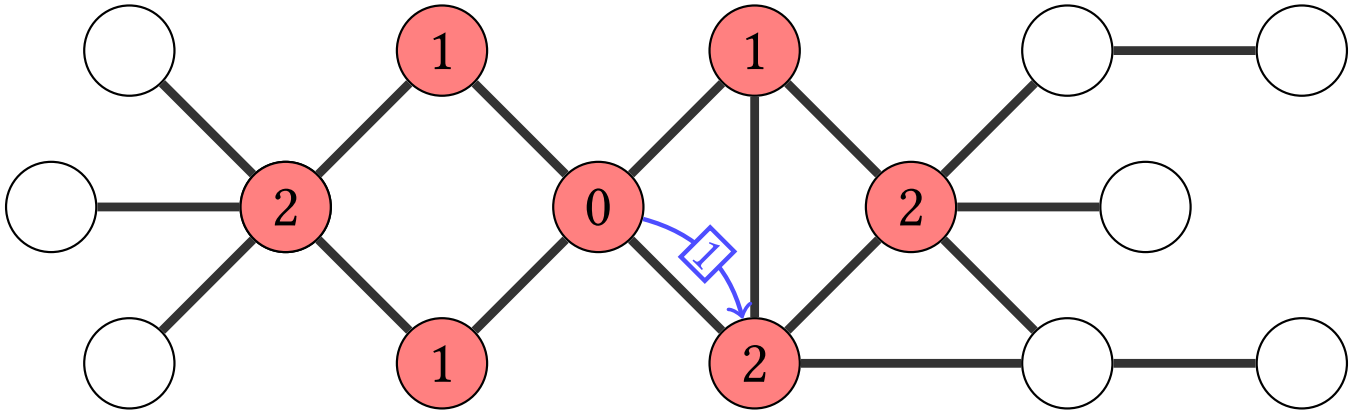
Beispiel: Breitensuche

Naive Ausführung synchroner Algorithmen ist in der Regel nicht erfolgreich (Inkonsistenzen)



Beispiel: Breitensuche

Naive Ausführung synchroner Algorithmen ist in der Regel nicht erfolgreich (Inkonsistenzen)



Pulsgeber

Ziel: Simuliere synchronen Algorithmus in asynchronem Netzwerk

Pulsgeber

Ziel: Simuliere synchronen Algorithmus in asynchronem Netzwerk

Zusätzlicher Pulsgeber-Algorithmus (**Synchronizer**)

Pulsgeber

Ziel: Simuliere synchronen Algorithmus in asynchronem Netzwerk

Zusätzlicher Pulsgeber-Algorithmus (**Synchronizer**)

- Generiert lokalen Puls für jeden Knoten
- Beim Puls i eines Knotens führt dieser Runde i aus: verarbeite alle bisher empfangenen Nachrichten und sende Nachrichten an Nachbarn
- Simulation ist korrekt wenn Puls $i + 1$ immer erst dann generiert wird, wenn Nachrichten von allen Nachbarn für Puls i empfangen wurden
- Achtung: Puls selbst ist nicht synchron für verschiedene Knoten

Pulsgeber

Ziel: Simuliere synchronen Algorithmus in asynchronem Netzwerk

Zusätzlicher Pulsgeber-Algorithmus (**Synchronizer**)

- Generiert lokalen Puls für jeden Knoten
- Beim Puls i eines Knotens führt dieser Runde i aus: verarbeite alle bisher empfangenen Nachrichten und sende Nachrichten an Nachbarn
- Simulation ist korrekt wenn Puls $i + 1$ immer erst dann generiert wird, wenn Nachrichten von allen Nachbarn für Puls i empfangen wurden
- Achtung: Puls selbst ist nicht synchron für verschiedene Knoten

Schwierigkeit: Knoten weiß nicht, von welchen Nachbarn er Nachrichten erhalten müsste

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben.

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Überprüfen der Safety:

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Überprüfen der Safety:

- Nach dem Empfang jeder Nachricht wird Bestätigungsnachricht an den jeweiligen Nachbarn gesendet

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Überprüfen der Safety:

- Nach dem Empfang jeder Nachricht wird Bestätigungsnachricht an den jeweiligen Nachbarn gesendet
- Zeit- und Nachrichtenkomplexität verdoppeln sich dadurch

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Überprüfen der Safety:

- Nach dem Empfang jeder Nachricht wird Bestätigungsnachricht an den jeweiligen Nachbarn gesendet
- Zeit- und Nachrichtenkomplexität verdoppeln sich dadurch

Beobachtung

Wenn für jeden Knoten immer Puls $i + 1$ erst dann generiert wird, sobald Knoten selbst und alle Nachbarn safe für Puls i sind, dann wird synchroner Algorithmus korrekt simuliert.

Safety

Definition

Ein Knoten ist *safe* für Puls i, wenn alle seine in der Simulation von Runde i gesendeten Nachrichten die jeweiligen Nachbarn erreicht haben. (Intuition: Simulation von Runde i ist für Knoten komplett abgeschlossen.)

Überprüfen der Safety:

- Nach dem Empfang jeder Nachricht wird Bestätigungsnachricht an den jeweiligen Nachbarn gesendet
- Zeit- und Nachrichtenkomplexität verdoppeln sich dadurch

Beobachtung

Wenn für jeden Knoten immer Puls $i + 1$ erst dann generiert wird, sobald Knoten selbst und alle Nachbarn safe für Puls i sind, dann wird synchroner Algorithmus korrekt simuliert.

$\rightarrow$ Unterschiedliche Strategien, um festzustellen, ob alle Nachbarn safe sind

Synchronizer α

Algorithmus:

- Sobald Knoten safe ist, sendet er eine safe-Nachricht an alle Nachbarn
- Knoten erkennen dadurch sofort, wenn alle Nachbarn safe sind
- Sobald Knoten und alle seine Nachbarn safe sind, wird neuer Puls für den Knoten generiert

Synchronizer α

Algorithmus:

- Sobald Knoten safe ist, sendet er eine safe-Nachricht an alle Nachbarn
- Knoten erkennen dadurch sofort, wenn alle Nachbarn safe sind
- Sobald Knoten und alle seine Nachbarn safe sind, wird neuer Puls für den Knoten generiert

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R)$ mit $O(M + Rm)$ Nachrichten simuliert werden.

Synchronizer α

Algorithmus:

- Sobald Knoten safe ist, sendet er eine safe-Nachricht an alle Nachbarn
- Knoten erkennen dadurch sofort, wenn alle Nachbarn safe sind
- Sobald Knoten und alle seine Nachbarn safe sind, wird neuer Puls für den Knoten generiert

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R)$ mit $O(M + Rm)$ Nachrichten simuliert werden.

Vorteil: Rein lokal, ohne Overhead in der Zeitkomplexität

Synchronizer α

Algorithmus:

- Sobald Knoten safe ist, sendet er eine safe-Nachricht an alle Nachbarn
- Knoten erkennen dadurch sofort, wenn alle Nachbarn safe sind
- Sobald Knoten und alle seine Nachbarn safe sind, wird neuer Puls für den Knoten generiert

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R)$ mit $O(M + Rm)$ Nachrichten simuliert werden.

Vorteil: Rein lokal, ohne Overhead in der Zeitkomplexität

Nachteil: Großer Overhead in der Nachrichtenkomplexität

Synchronizer β

Idee: Puls wird global vom Leader generiert

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet
- Beim Erhalt von Puls i führen Knoten Runde i aus

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet
- Beim Erhalt von Puls i führen Knoten Runde i aus
- Knoten ist safe, sobald er alle Empfangsbestätigungen erhalten hat

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet
- Beim Erhalt von Puls i führen Knoten Runde i aus
- Knoten ist safe, sobald er alle Empfangsbestätigungen erhalten hat
- Neuer Puls wird von Leader generiert sobald alle Knoten safe sind

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet
- Beim Erhalt von Puls i führen Knoten Runde i aus
- Knoten ist safe, sobald er alle Empfangsbestätigungen erhalten hat
- Neuer Puls wird von Leader generiert sobald alle Knoten safe sind
- Leader erfährt durch aggregierten Upcast im Baum, wenn alle Knoten safe sind

Synchronizer β

Idee: Puls wird global vom Leader generiert

- Breitensuchbaum mit Leader als Wurzel
- Jeder Puls wird per Downcast im Baum an alle Knoten gesendet
- Beim Erhalt von Puls i führen Knoten Runde i aus
- Knoten ist safe, sobald er alle Empfangsbestätigungen erhalten hat
- Neuer Puls wird von Leader generiert sobald alle Knoten safe sind
- Leader erfährt durch aggregierten Upcast im Baum, wenn alle Knoten safe sind

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann im asynchronen Netzwerken in Zeit $O(RD)$ mit $O(M + Rn)$ Nachrichten simuliert werden, wenn bereits ein Leader und ein Breitensuchbaum berechnet wurden.

Asynchrone Breitensuche

Achtung: Synchronizer β benötigt Breitensuchbaum

Asynchrone Breitensuche

Achtung: Synchronizer β benötigt Breitensuchbaum

Asynchrone Berechnung eines Breitensuchbaums:

- Synchroner Algorithmus: $O(D)$ Runden, $O(m)$ Nachrichten
  Mit Synchronizer α: $O(D)$ Zeiteinheiten, $O(mD)$ Nachrichten
- Alternative: $O(D^2)$ Zeiteinheiten, $O(m + nD)$ Nachrichten

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Definition

Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Definition

Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:

1. Jede Teilmenge P_i hat ein designiertes Zentrum und die Knoten in P_i sind einem Baum mit Entfernung höchstens ρ vom Zentrum organisiert.

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Definition

Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:

1. Jede Teilmenge P_i hat ein designiertes Zentrum und die Knoten in P_i sind einem Baum mit Entfernung höchstens ρ vom Zentrum organisiert.
2. Es wurde eine Menge F von höchstens μ Kanten ausgewählt so dass
   1. jede Kante in F benachbarte Teilmengen P_i und P_j verbindet und
   2. für jedes Paar benachbarter Teilmengen mindestens eine Kante in F enthalten ist.

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Definition

Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:

1. Jede Teilmenge P_i hat ein designiertes Zentrum und die Knoten in P_i sind einem Baum mit Entfernung höchstens ρ vom Zentrum organisiert.
2. Es wurde eine Menge F von höchstens μ Kanten ausgewählt so dass
   1. jede Kante in F benachbarte Teilmengen P_i und P_j verbindet und
   2. für jedes Paar benachbarter Teilmengen mindestens eine Kante in F enthalten ist.

Ideal: Maximal eine ausgewählte Kante zwischen benachbarten Teilmengen

Aber: Schwer zu berechnen im CONGEST Modell

Hybride Synchronisierung

Idee: Hybrid zwischen lokalem und globalem Puls

Definition

Eine (ρ, μ)-Partition ist eine Zerlegung der Knoten V in nicht-leere, disjunkte Teilmengen $P_1, \ldots, P_\ell$ mit folgenden Eigenschaften:

1. Jede Teilmenge P_i hat ein designiertes Zentrum und die Knoten in P_i sind einem Baum mit Entfernung höchstens ρ vom Zentrum organisiert.
2. Es wurde eine Menge F von höchstens μ Kanten ausgewählt so dass
   1. jede Kante in F benachbarte Teilmengen P_i und P_j verbindet und
   2. für jedes Paar benachbarter Teilmengen mindestens eine Kante in F enthalten ist.

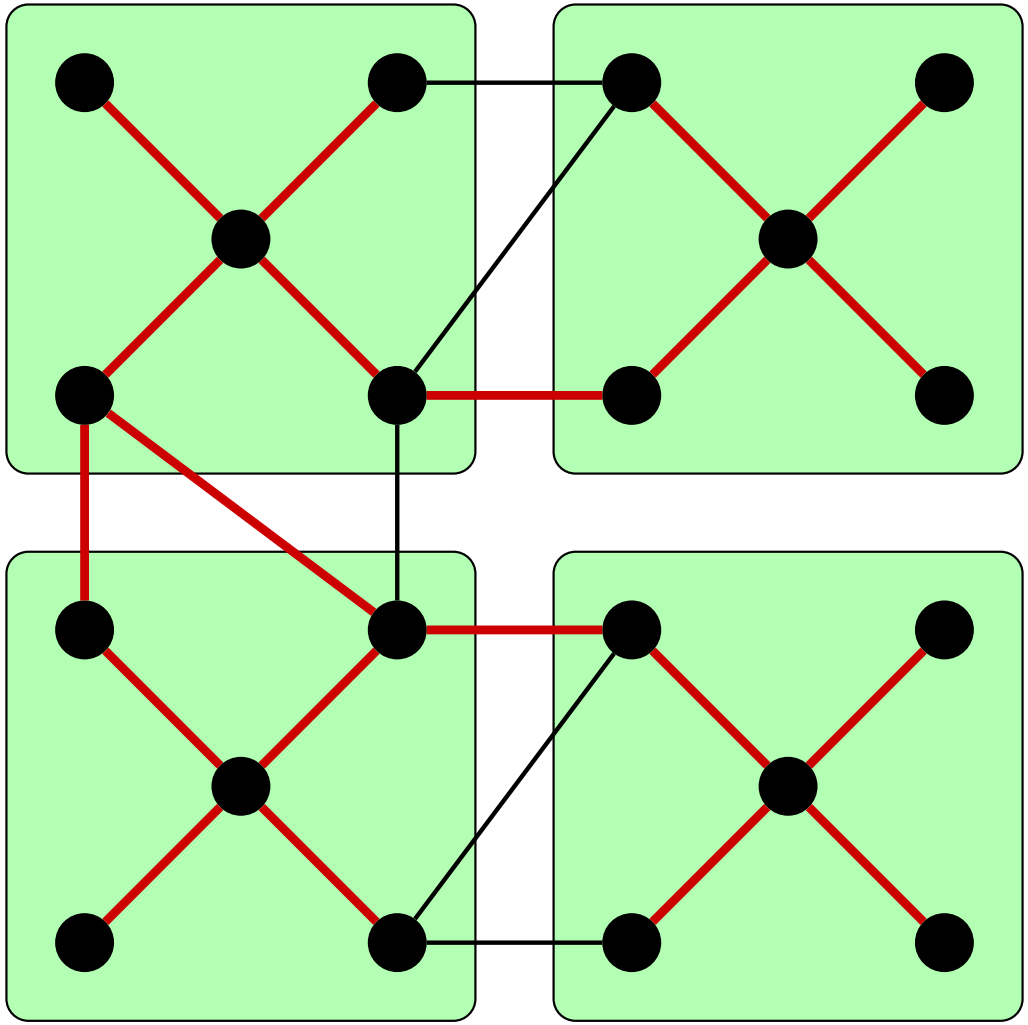
Ideal: Maximal eine ausgewählte Kante zwischen benachbarten Teilmengen

Aber: Schwer zu berechnen im CONGEST Modell

Vorgehensweise:

- Synchronizer α zwischen den Teilmengen
- Synchronizer β innerhalb der Teilmengen

Beispiel für (ρ, μ)-Partition



(ρ, μ)-Partition mit $\rho = 1$ und $\mu = 4$

Berechnung einer Partition

Theorem ([Elkin/Neiman '17] [Forster/Grösbacher/de Vos '21])

Im synchronen CONGEST Modell kann für jede Ganzzahl $k \geq 2$ eine (ρ, μ)-Partition mit $\rho = k - 1$ und $\mu = O(n^{1+1/k})$ in Erwartung in $O(k)$ Runden berechnet werden.

Berechnung einer Partition

Theorem ([Elkin/Neiman '17] [Forster/Grösbacher/de Vos '21])

Im synchronen CONGEST Modell kann für jede Ganzzahl $k \geq 2$ eine (ρ, μ)-Partition mit $\rho = k - 1$ und $\mu = O(n^{1+1/k})$ in Erwartung in $O(k)$ Runden berechnet werden.

Mit Synchronizer α:

Korollar

Im asynchronen CONGEST Modell kann für jede Ganzzahl $k \geq 2$ eine (ρ, μ)-Partition mit $\rho = k - 1$ und $\mu = O(n^{1+1/k})$ in Erwartung in Zeit $O(k)$ mit $O(km)$ Nachrichten berechnet werden.

Synchronizer γ

Algorithmus:

- Synchronisierung innerhalb einer Teilmenge wie bei Synchronizer β:

Synchronizer γ

Algorithmus:

- Synchronisierung innerhalb einer Teilmenge wie bei Synchronizer β:
  - Jedes Zentrum einer Teilmenge generiert Puls für alle Knoten der Teilmenge
  - Downcast des Pulses im Baum vom Zentrum aus zu allen Knoten der Teilmenge
  - Knoten simuliert aktuelle Runde nach Erhalt des Pulses
  - Knoten ist safe sobald er alle Empfangsbestätigungen erhalten hat
  - Teilmenge ist safe, wenn alle Knoten der Teilmenge safe sind
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn Teilmenge safe ist

Synchronizer γ

Algorithmus:

- Synchronisierung innerhalb einer Teilmenge wie bei Synchronizer β:
  - Jedes Zentrum einer Teilmenge generiert Puls für alle Knoten der Teilmenge
  - Downcast des Pulses im Baum vom Zentrum aus zu allen Knoten der Teilmenge
  - Knoten simuliert aktuelle Runde nach Erhalt des Pulses
  - Knoten ist safe sobald er alle Empfangsbestätigungen erhalten hat
  - Teilmenge ist safe, wenn alle Knoten der Teilmenge safe sind
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn Teilmenge safe ist
- Synchronisierung zwischen den Teilmengen wie bei Synchronizer α:

Synchronizer γ

Algorithmus:

- Synchronisierung innerhalb einer Teilmenge wie bei Synchronizer β:
  - Jedes Zentrum einer Teilmenge generiert Puls für alle Knoten der Teilmenge
  - Downcast des Pulses im Baum vom Zentrum aus zu allen Knoten der Teilmenge
  - Knoten simuliert aktuelle Runde nach Erhalt des Pulses
  - Knoten ist safe sobald er alle Empfangsbestätigungen erhalten hat
  - Teilmenge ist safe, wenn alle Knoten der Teilmenge safe sind
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn Teilmenge safe ist
- Synchronisierung zwischen den Teilmengen wie bei Synchronizer α:
  - Sobald Teilmenge safe ist, werden benachbarte Teilmengen darüber informiert:
    - ★ Downcast der Information vom Zentrum aus in eigener Teilmenge
    - ★ Knoten in eigener Teilmenge informieren benachbarte Teilmengen über ausgewählte Kanten zu Nachbarn in diesen Teilmengen
  - Zentrum erfährt durch aggregierten Upcast im Baum, wenn über alle ausgewählten Kanten zu benachbarten Teilmengen Information erhalten wurde, dass entsprechende Teilmenge safe ist

Garantien

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R\rho)$ mit $O(M + R(\mu + n))$ Nachrichten simuliert werden, wenn bereits eine (ρ, μ)-Partition berechnet wurde.

Garantien

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R\rho)$ mit $O(M + R(\mu + n))$ Nachrichten simuliert werden, wenn bereits eine (ρ, μ)-Partition berechnet wurde.

Mit Algorithmus zur Berechnung einer Partition:

Korollar

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(Rk)$ mit in Erwartung $O(M + Rn^{1+1/k} + mk)$ Nachrichten simuliert werden.

Garantien

Theorem

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(R\rho)$ mit $O(M + R(\mu + n))$ Nachrichten simuliert werden, wenn bereits eine (ρ, μ)-Partition berechnet wurde.

Mit Algorithmus zur Berechnung einer Partition:

Korollar

Jeder synchrone Algorithmus, der R Runden benötigt und dabei M Nachrichten versendet, kann in asynchronen Netzwerken in Zeit $O(Rk)$ mit in Erwartung $O(M + Rn^{1+1/k} + mk)$ Nachrichten simuliert werden.

Insbesondere: Zeit $O(R \log n)$ mit $O(M + Rn + m \log n)$ Nachrichten für $k = \lceil \log n \rceil$

Zusammenfassung

	Zeit	**Nachrichten**	**Initialisierungsaufwand**
Synchronizer α	$O(R)$	$O(M + Rm)$	–
Synchronizer β	$O(RD)$	$O(M + Rn)$	Breitensuchbaum
Synchronizer γ	$O(R\rho)$	$O(M + R(\mu + n))$	(ρ, μ)-Partition

Zusammenfassung

	Zeit	**Nachrichten**	**Initialisierungsaufwand**
Synchronizer α	$O(R)$	$O(M + Rm)$	–
Synchronizer β	$O(RD)$	$O(M + Rn)$	Breitensuchbaum
Synchronizer γ	$O(R\rho)$	$O(M + R(\mu + n))$	(ρ, μ)-Partition

Rechtfertigt Einschränkung auf synchrone Algorithmen!

Quellen

Der Inhalt dieser Vorlesungseinheit basiert zum Teil auf einer Vorlesungseinheit von Christoph Lenzen.

Literatur:

- Baruch Awerbuch. „Complexity of Network Synchronization“. *Journal of the ACM* 32(4): 804–823 (1985)
- Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, Michael E. Saks. „Adapting to Asynchronous Dynamic Networks“. In: *Proc. of the Symposium on Theory of Computing (STOC)*. 1992, S. 557–570
- Michael Elkin, Ofer Neiman. „Efficient Algorithms for Constructing Very Sparse Spanners and Emulators“. *ACM Transactions on Algorithms* 15(1): 4:1–4:29 (2019)
- Sebastian Forster, Martin Grösbacher, Tijn de Vos. „An Improved Random Shift Algorithm for Spanners and Low Diameter Decompositions“. In: *Proc. of the International Conference on Principles of Distributed System (OPODIS)*. 2021, S. 16:1–16:17
- David Peleg (2000) *Distributed Computing*, Kapitel 6, SIAM.