Asynchronous Network Computing with Process Failures

Tim D'Avis

- Einleitung
- Randomization
- Failure Detectors
- k-Agreement
- Approximate Agreement

Einleitung - 1

- Netzwerk Modell:
  - Asynchronous broadcast system
    - nur stopping failures
      (modelliert durch stop events)
    - channels sind zuverlässig
      (kein Nachrichtenverlust, usw.)

Einleitung - 2

- Auch möglich:
  Asynchronous send/receive system
  - A broadcast system
  - B send/receive system
  - B simuliert A so, daß ein User U die Ausführung von A und B nicht unterscheiden kann

Einleitung - 3

- Process P_i
  - input actions:
    - $init(v)_i$
    - $stop_i$
  - output actions:
    - $decide(v)_i$
- User U_i
  - output actions:
    - $init(v)_i$
  - input actions:
    - decide(v)

$v \epsilon V$, V Nachrichenmenge
$i \epsilon \{1,...,n\}$
Alle Aktionen mit Subscript i geschehen an port i

Einleitung - 4

Well-formedness:
In any execution, and for any i, the interactions between U_i and A are well-formed for i.

Agreement:
In any execution, all decision values are identical.

Validity:
In any execution, if all init actions that occur contain the same value v, then v is the only possible decision value.

Einleitung - 5

Failure-free termination:
In any fair failure-free execution in which init events occur on all ports, a decide event occurs on each port.

An asynchronous network system *solves the agreement problem* if it guarantees: agreement, validity and failure-free termination (for all collections of users).

Einleitung - 6

f-failure termination, $0 \leq f \leq n$:
In any fair execution in which init events occur on all ports, if there are stop events on at most f ports, then a decide event occurs on every non-failing port.

Wait-free termination
is f-failure termination where f = n

Einleitung - 7

Theorem 21.2

There is no algorithm in the asynchronous broadcast model with a reliable broadcast channel that solves the agreement problem and guarantees 1-failure termination.

Einleitung - 8

- Wege um dennoch einen korrekten Algorithmus zu erhalten
  - Randomization, durch zulassen von random choices
  - Model um failure detector erweitern
  - Agreement auf Menge von Werten
  - Agreement mit Toleranz ε

Randomization - 1

- Modell wird verstärkt, denn es erlaubt *random choices* während der Berechnung
- Korrektheitsbedingungen werden abgeschwächt
  - well-formedness, agreement, validity unverändert
  - termination nur noch mit gewisser Wahrscheinlichkeit
    - All the faulty processes will decide by time t after the arrival of all inputs, with probability at least p(t), where p is a particular monotone non-decreasing, unbounded function.

Randomization - 2

n > 3f, V = {0,1}

***BenOr* algorithm:**

Each process P_i has local variables x and y, initially *null*. An $init(v)_i$ input causes process P_i to set $x := v$. P_i executes a series of *stages* numbered $1, 2, \ldots$, each stage consisting of two *rounds*. P_i begins stage 1 after it receives its initial value in an $init_i$ input. It continues performing the algorithm forever, even after it decides.

At each stage $s \geq 1$, P_i does the following:

Round 1: P_i broadcasts ("*first*", s, v), where v is its current value of x, then waits to obtain $n - f$ messages of the form ("*first*", $s, *$). If all of these have the same value v, then P_i sets $y := v$; otherwise it sets $y := null$.

Round 2: P_i broadcasts ("*second*", s, v), where v is its current value of y, then waits to obtain $n - f$ messages of the form ("*second*", $s, *$). There are three cases: First, if all of these have the same value $v \neq null$, then P_i sets $x := v$ and performs a $decide(v)_i$ if it has not already done so. Second, if at least $n - 2f$ of these messages, but not all of the messages, have the same value $v \neq null$, then P_i sets $x := v$ (but does not decide). (The assumption that $n > 3f$ implies that there cannot be two different such values v.) Otherwise, P_i sets x to either 0 or 1, choosing randomly with equal probability.

Failure Detectors - 1

- Modell wird ein failure detector hinzugefügt
- failure detector liefert Informationen über alle stoppenden Prozesse
- Hier: Perfekter failure detector, d.h. Angaben sind immer korrekt (lügt nicht, macht keine falschen Annahmen)

Failure Detectors - 2

- Prozesse werden erweitert
  - zusätzliche Inputs:
    - $stop_i$ um gestoppt zu werden
    - $informed\text{-}stopped(j)_i$ um zu erfahren das j gestoppt hat

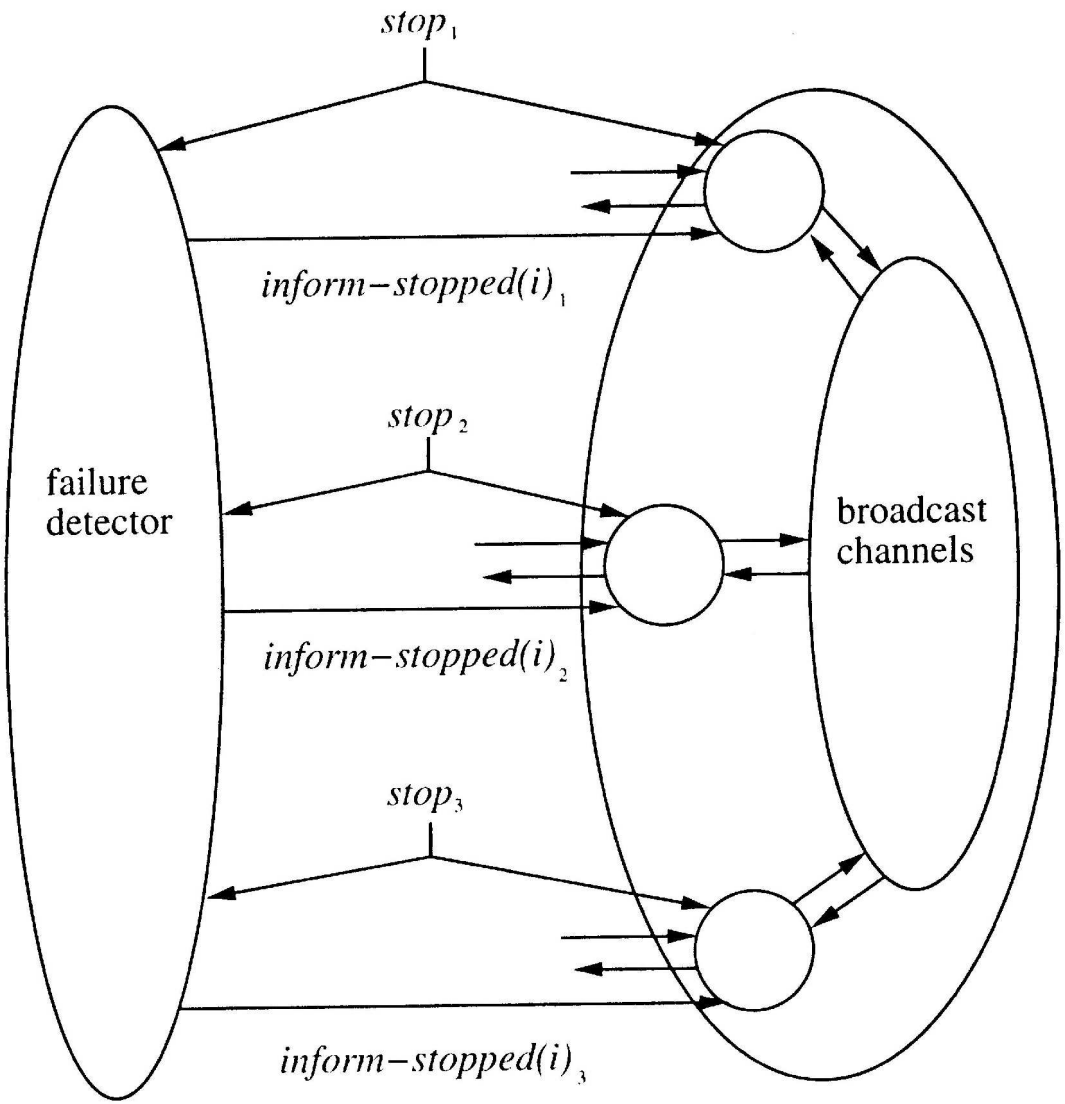

Figure 21.1: Architecture for asynchronous broadcast system with a perfect failure detector.

Failure Detecors - 3

- Jeder Prozess P_i hat zwei Variablen die er zu stabilisieren versucht
  - Einen Vektor *val,* welcher für alle Prozesse {1,...,n} deren bisher bekannten Anfangswerte speichert
  - Eine Menge *stopped* die alle Prozesse enthält von denen i bereits erfahren hat das sie gestoppt haben
- P_i broadcasted ständig val und stopped und aktualisiert diese sobald es neue Informationen von einem anderen Prozess lernt

Failure Detecors - 4

- P_i „ratifiziert“ ständig seine Daten, indem es seine mit denen der anderen Prozesse vergleicht
- Haben sich val und stopped „stabilisiert“, dann entscheidet P_i auf den kleinsten Wert in val der nicht null ist

PerfectFDAgreement$_i$ automaton (formal):

Signature:

Input:
- $init(v)_i$, $v \in V$
- $receive(w, I)_{j,i}$, $w \in W$, $I \subseteq \{1, \ldots, n\}$, $1 \leq j \leq n$
- $inform\text{-}stopped(j)_i$, $j \neq i$

Output:
- $bcast(w, I)_i$, $w \in W$, $I \subseteq \{1, \ldots, n\}$
- $decide(v)_i$, $v \in V$

States:

$val \in W$, initially identically *null*
$stopped \subseteq \{1, \ldots, n\}$, initially $\emptyset$
$ratified \subseteq \{1, \ldots, n\}$, initially $\emptyset$
decided, a Boolean, initially *false*

Transitions:

$init(v)_i$
- Effect:
  - $val(i) := v$
  - $ratified := \{i\}$

$inform\text{-}stopped(j)_i$
- Effect:
  - $stopped := stopped \cup \{j\}$
  - $ratified := \{i\}$

$bcast(w, I)_i$
- Precondition:
  - $w = val$
  - $I = stopped$
  - $val(i) \neq null$
- Effect:
  - none

$receive(w, I)_{j,i}$
- Effect:
  - if $j \notin stopped$ then
    - if $(w, I) = (val, stopped)$ then
      - $ratified := ratified \cup \{j\}$
    - else if $(w, I) \not\leq_d (val, stopped)$ then
      - $stopped := stopped \cup I$
      - for all k, $1 \leq k \leq n$, do
        - if $val(k) = null$ then $val(k) := w(k)$
      - $ratified := \{i\}$

$decide(v)_i$
- Precondition:
  - $ratified \cup stopped = \{1, \ldots, n\}$
  - $v = val(j)$, where j is the smallest index with $v(j) \neq null$
  - $decided = false$
- Effect:
  - $decided := true$

Tasks:

$\{bcast(w, I)_i : w \in W, I \subseteq \{1, \ldots, n\}\}$
$\{decide(v)_i : v \in V\}$

k-Agreement - 1

- $f < k$
  - f Anz. Fehler
  - k Anz. Entscheidungswerte, $k \geq 1$
- Schwächung des problem statements
  - well-formedness, termination unverändert
  - Agreement: In any execution, there is a subset W of V, $|W| = k$, such that all decision values are in W.
  - Validity: In any execution, any decision value for any process is the inital value of some process.

k-Agreement - 2

- Einfacher Algorithmus:
  - TrivialKAgreement algorithm:
    Processes P_1, P_2, ... P_k (only) broadcast their inital values.
    Every process P_i decides on the first value it receives.

Approximate Agreement - 1

- Menge V = R
- Prozesse können reelle Werte verschicken
- Einigung nicht auf exakten Wert, sondern mit Toleranz ε ($\varepsilon > 0$)
- Schwächung des problem statements
  - well-formedness, termination unverändert
  - Agreement: In any execution, any two values are within ε of each other.
  - Validity: In any execution, any decision value is within the range of the initial values.

Approximate Agreement - 2

- n > 3f
- AsynchApproxAgreement algorithm:
  Each P_i maintains a variable *val* containing its latest estimate. This gets initalized to the value *v* that arrives in an init(v)$_i$ input. At each stage, P_i does the following:
  - It broadcasts *val*, tagged with the stage number *s*.
  - Then it collects the first *n - f* values it receives for stage *s* into a multiset *W*.
  - Finally, it sets *val* to mean(select(W)).

Approximate Agreement - 3

- Terminierung:
  - spezielle Initalisierungs stage 0
    - broadcast val
    - collect n - f Values in W
    - val = median(W)
  - Berechne aus diesem val, die stage number i in der sicher ist das alle decision values höchstens ε auseinander liegen
  - Nach stage i verschicke halt-tag. Alle Prozesse verwenden von da an für diesen Prozess den letzten erhaltenen Wert.