Simulation von verteilten Algorithmen im FAR-Modell

Verteidigung Großer Beleg

Josef Spillner <js177634@inf.tu-dresden.de>

12.01.2006

Aufbau des Vortrags

0) Motivation
1) wichtige Grundlagen
2) Einführung in das FAR-Modell
3) Präsentation der Software
4) experimentelle Nachweise und deren Relevanz

Motivation

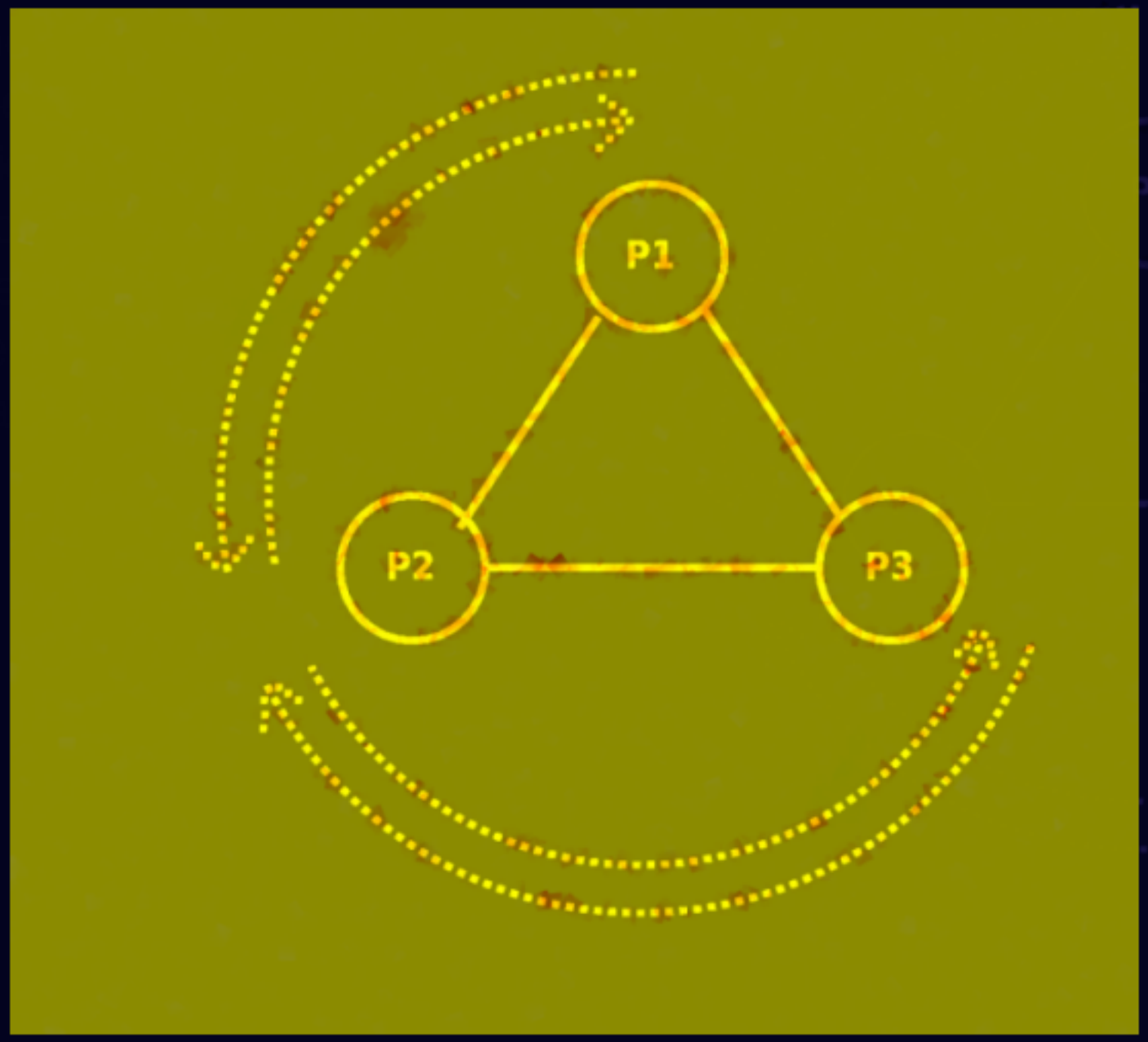
Verteilte Systeme: grau ist alle Theorie...

- welche Parameter sind optimal?
- welche Ereignisse können unerwartet auftreten (z.B. Deadlock)?

FAR-Modell:

- bisher noch keine Software-Umsetzung
- automatisierbare Vergleiche mit anderen Modellen

Teil 1: Grundlagen



Grundlagen: Systemaufbau

Primitiven: Prozesse, Kanäle (statisch), Nachrichten (dynamisch)

Prozesse:
- autonome Einheiten in einem System
- können nur durch Kommunikation kooperieren (kein SHM etc.)

Kanäle:
- übertragen Nachrichten
- existieren jeweils zwischen zwei Prozessen
- hier betrachtet: vollständige Verbundenheit

Nachrichten:
- enthalten Informationen
- oft mit Header versehen (z.B. Sequenznummer)

Grundlagen: verteilte Algorithmen

Prozesse, die miteinander interagieren
- dynamisch: hinzukommende und ausfallende Prozesse
- Abstimmungen, Überprüfungen
- Redundanz, Replikation, Sicherheit

- können fehlerhaft sein (endgültiger Absturz)
  - byzantinische Fehler, Wiederbelebung etc. hier nicht betrachtet

Beteiligte Kanäle
- können Nachrichten verzögern oder verlieren

Algorithmen: sehr viele, inklusive Varianten
- Leader Election
- Atomic Broadcast
- {1,2,3}-Phasen-Commit-Protokoll
- Consensus <-------

Grundlagen: Consensus

Consensus:

- Zielstellung:
  - Ermittlung eines global bekannten Wertes
  - Konsens unter allen korrekten Teilnehmern
- Protokoll:
  - n Teilnehmer (p[0..n-1]), vollständig miteinander verbunden
  - Ausführung in asynchronen Runden
  - Einigung auf einen Wert durch Nachrichtenaustausch
  - in Runde r: Koordinator = p[n % r]
  - > 50% aller Prozesse (> n/2) müssen korrekt sein

Eigenschaften des Consensus:

- Validität: der gefundene Wert wurde von einem Teilnehmer vorgeschlagen
- Terminierung: der Algorithmus wird beendet, es wird ein Wert gefunden
- Übereinkunft: keine zwei Prozesse entscheiden gegensätzlich

Grundlagen: Synchronität

Synchrones System:

- Methode zum Versenden von Nachrichten liefert Rückgabewert
  - erfolgreich: Nachricht ist beim Empfänger angekommen + bestätigt
  - fehlerhaft: Nachricht ist verloren bzw. Timeout
- erfordert entsprechendes Kanalmodell (reliable, ...)

Asynchrones System:

- es existiert kein Timeout
  - weder für Versand noch für Bearbeitung/Bestätigung
- ist z.B. über UDP implementierbar

Partiell synchrone System:

- wichtige Variablen: Φ und Δ
  - Φ: Bearbeitungszeit einer Nachricht in einem Prozess
  - Δ: Laufzeit einer Nachricht in einem Kanal
- entweder Φ oder Δ existieren (sind aber unbekannt)
- alternativ: Φ oder Δ sind bekannt (treten aber erst irgendwann ein)
  - nach globaler Stabilisierungszeit GST

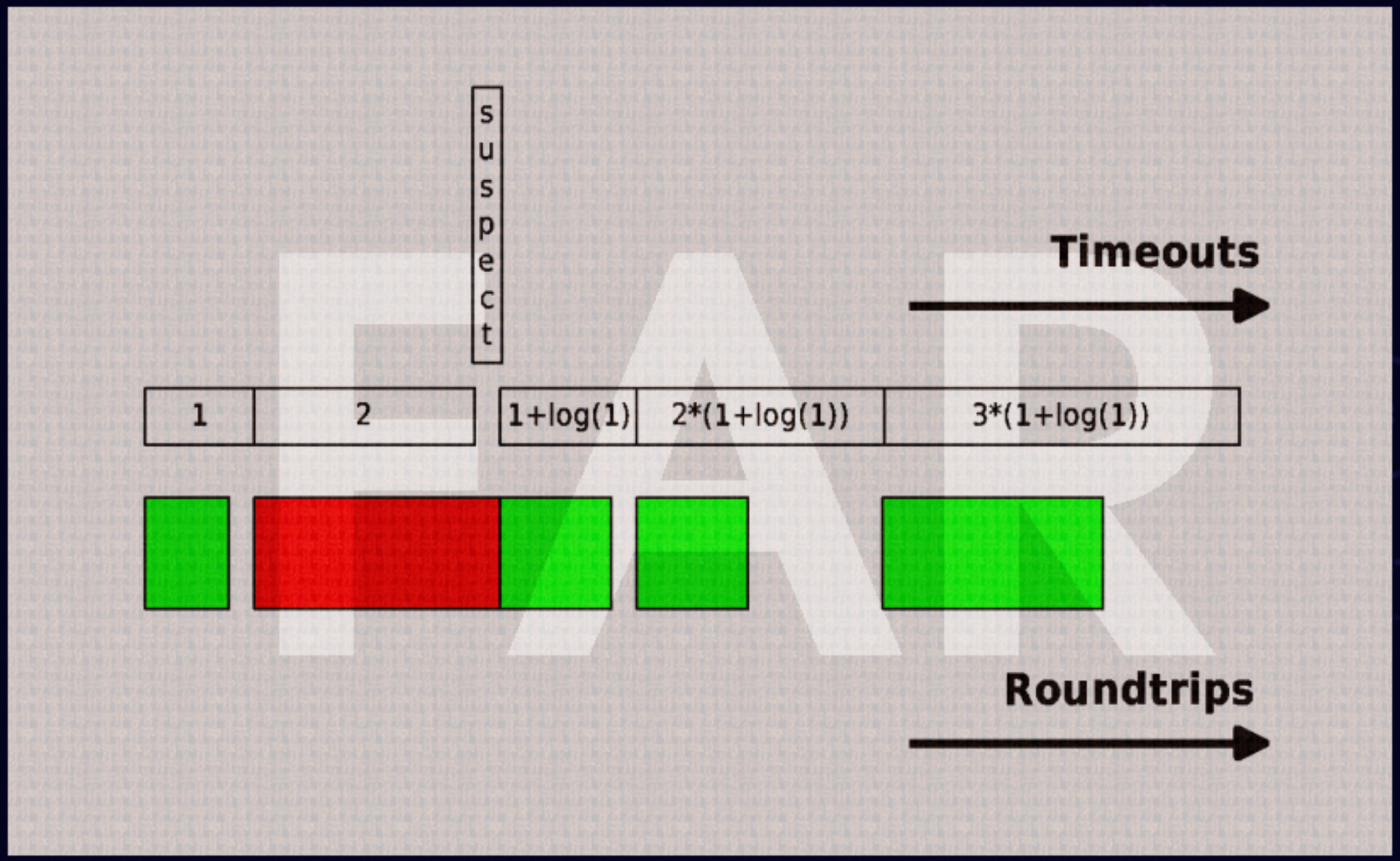
Grundlagen: Fehlererkennung

Bekannt:
- in vollständig asynchronen Systemen kein Consensus möglich
  - sofern auch nur ein Prozess abstürzen darf

Einführung von Fehlerdetektoren (FD):
- sind einem Prozess zugeordnete Module
- können andere Prozesse verdächtigen
- sind (zumindest in der Praxis) unzuverlässig (◊-Notation)
  - ihre Charakteristik stellt sich erst irgendwann ein
  - Verdächtigung sollte Genauigkeit und Vollständigkeit aufweisen
  - mehrere Basisklassen (schwache und starke FD)

Im partiell synchronen Modell:
- PS-FD sendet periodisch ALIVE-Nachrichten
- Empfänger stuft Sender entsprechend eines Timeouts ein

Teil 2: Einführung in FAR



Vorstellung: FAR-Modell

Es werden wenige Annahmen gemacht
- zugeschnitten auf physisch vorhandene Infrastruktur

Es werden Stubborn Channels verwendet
- Nachricht erreicht letztlich Empfänger, solange keine weitere gesendet wird
  - Nachricht wird immer wieder durch Kanal verschickt
- Es können Nachrichten verloren gehen, je nach zugrundeliegender Schicht
- UDP ist für die Implementierung geeignet

Es wird ein Fehlerdetektor „EA-FD" ermöglicht
- vergleichbar mit anderen FDs der Kategorie $\Diamond\mathcal{P}$
  - letztendlich perfekter Fehlerdetektor
  - irgendwann werden alle korrekten Prozesse alle fehlerhaften verdächtigen
  - und keiner von ihnen wird einen korrekten Prozess verdächtigen
- Berechnung der Verdächtigungszeit aus Zahl der langsamen/schnellen Nachrichten

FAR-Modell (formal)

Annahmen:

- die mittlere Antwortzeit ist endlich
  - siehe reales Zeitverhalten im Internet: keine GST
- es existiert eine schwache Uhr
  - Inkrementierung eines Ganzzahlwertes

EA-FD:

- timeout = (1 + #fastmsgs) * (1 + log(1 + #slowmsgs))

Teil 3: Software-Implementierung

```
0.431180000305 <Channel <node#3>-<node#2>> sending acknowledgement [(13, 'ACK:
('decide', (0, 5), 80) FROM node#3 IN node#2 <Round: (0, 4)>
0.432481050491 (sender node#4:s) received message (ack) 10 from node#2
433337392807 (sender node#4:s) EA-FD: 2 fast, 0 slow, diff = 0.126781225204, time
(channel) ack read after sending new message
0.434764862061 <Channel <node#4>-<node#2>> got message [(15, ('decide', (0, 5)
0.43551492691 <Channel <node#4>-<node#2>> sending acknowledgement [(15, 'ACK:
('decide', (0, 5), 80) FROM node#4 IN node#2 <Round: (0, 4)>
0.436820983887 (sender node#4:s) received message (ack) 16 from node#3
437736988068 (sender node#4:s) EA-FD: 1 fast, 0 slow, diff = 0.0174689292908, tim
438500881195 (sender node#3:s) received message (ack) 14 from node#4
439428091049 (sender node#3:s) EA-FD: 1 fast, 0 slow, diff = 0.110801935196, time
# Scheduler information:
# 3 channels
<Channel <node#3>-<node#2>> (stubborn/internal)
(node#3->node#2): active
<Channel <node#4>-<node#2>> (stubborn/internal)
(node#4->node#2): active
# <Channel <node#4>-<node#3>> (stubborn/internal)
# (node#4->node#3): inactive
3 nodes
node#2 [FD: ea] (idle)
# [s] node#2:s: 2 messages pending 2 done
# [r] receiver#1: local
```

Existierende Software oder Neuentwicklung

Es existieren bereits diverse Simulationsumgebungen:

NS2
- Simulation mit Tcl-Skripten
- „Emulation” (verteilte Ausführung) ist experimentell vorhanden

OMNeT++
- Einsatz von NED-Dateien (Network Description)
- keine verteilte Ausführung möglich

Consensus auf L4
- C++-Implementierung unter Nutzung der FIPS-Netzwerkanbindung
- auf Anfrage von Tiago Jorge erhältlich
- siehe Mailingliste l4-hackers auf os.inf.tu-dresden.de, Februar 2005 ff

Neuentwicklung nicht abwegig, um reale Messungen durchzuführen!
- Algorithmen mussten ohnehin von Grund auf implementiert werden
- und unbestreitbar höhere Identifikation mit der Thematik

Software-Neuentwicklung

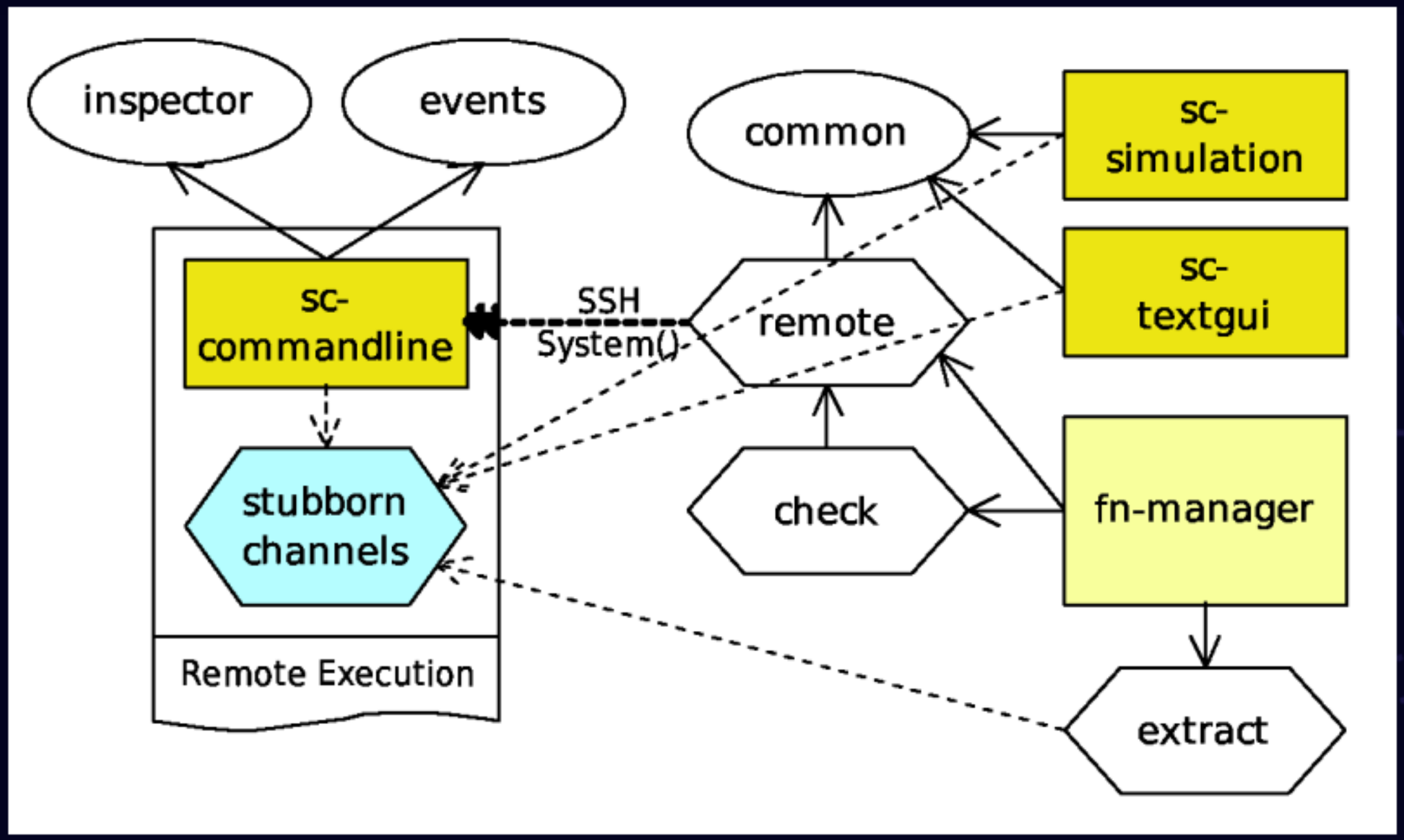
FARnodes: verteilte Ausführung und Simulation

- Ausführung: reelle Nachrichtenübermittlung
  - z.B. auf Basis von UDP
- Simulation: nur interne Objektkommunikation
  - möglich auf der Basis reeller Zeitwerte (Traces)

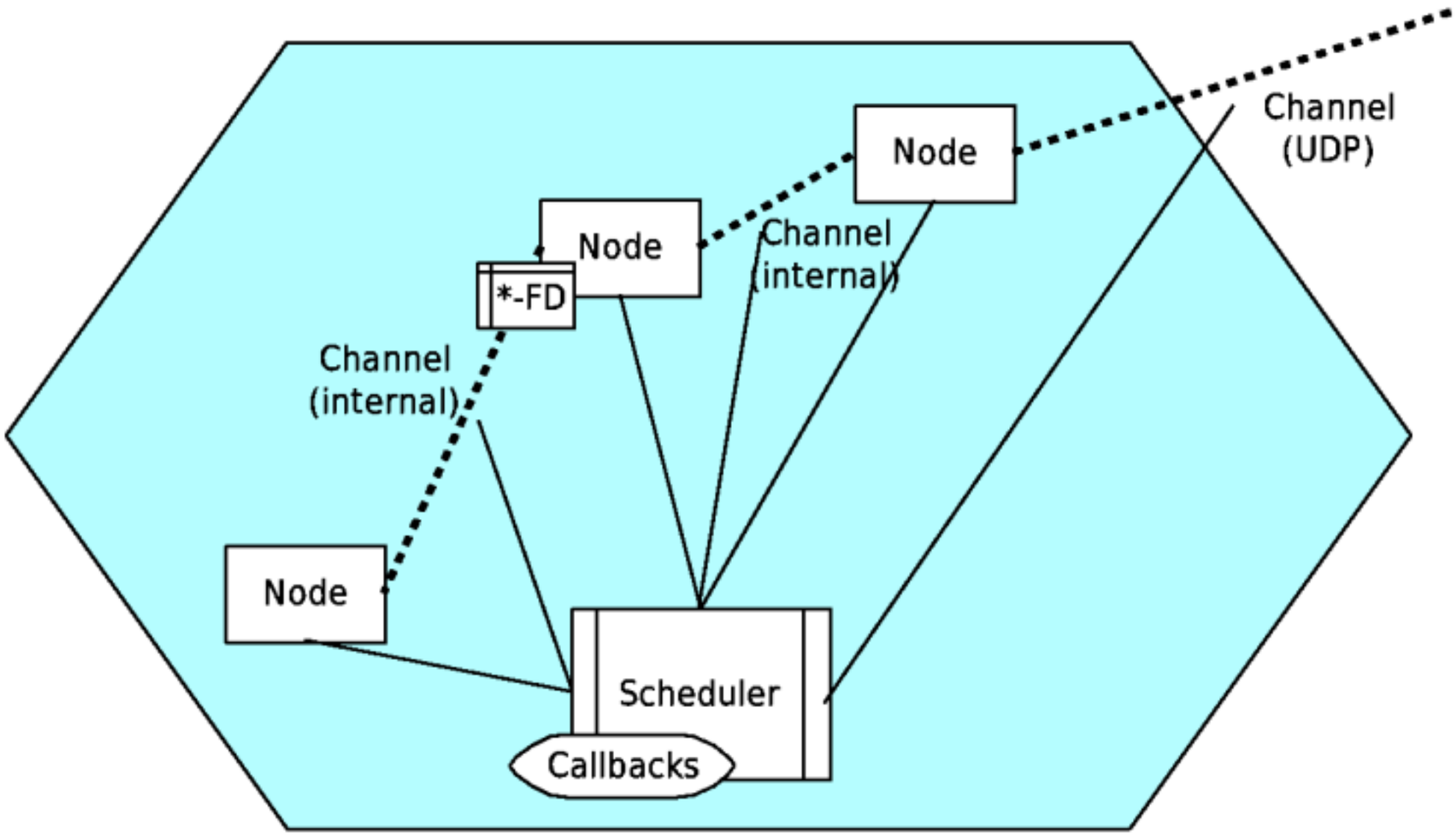
- Implementierung in Python
  - plattformunabhängig, interpretiert -> schnelle Testzyklen
  - Auslagerung kritischer Routinen nach C++ wäre möglich (z.B. Initialisierung)

- etwa 4000 Zeilen Code
  - fehlende Dokumentation in den Sourcen, aber:
  - API-Docs, Nutzerdokumentation, Diagramme mit Beschreibungen

FARnodes-Architektur



Kommunikation



Nutzung von FARnodes

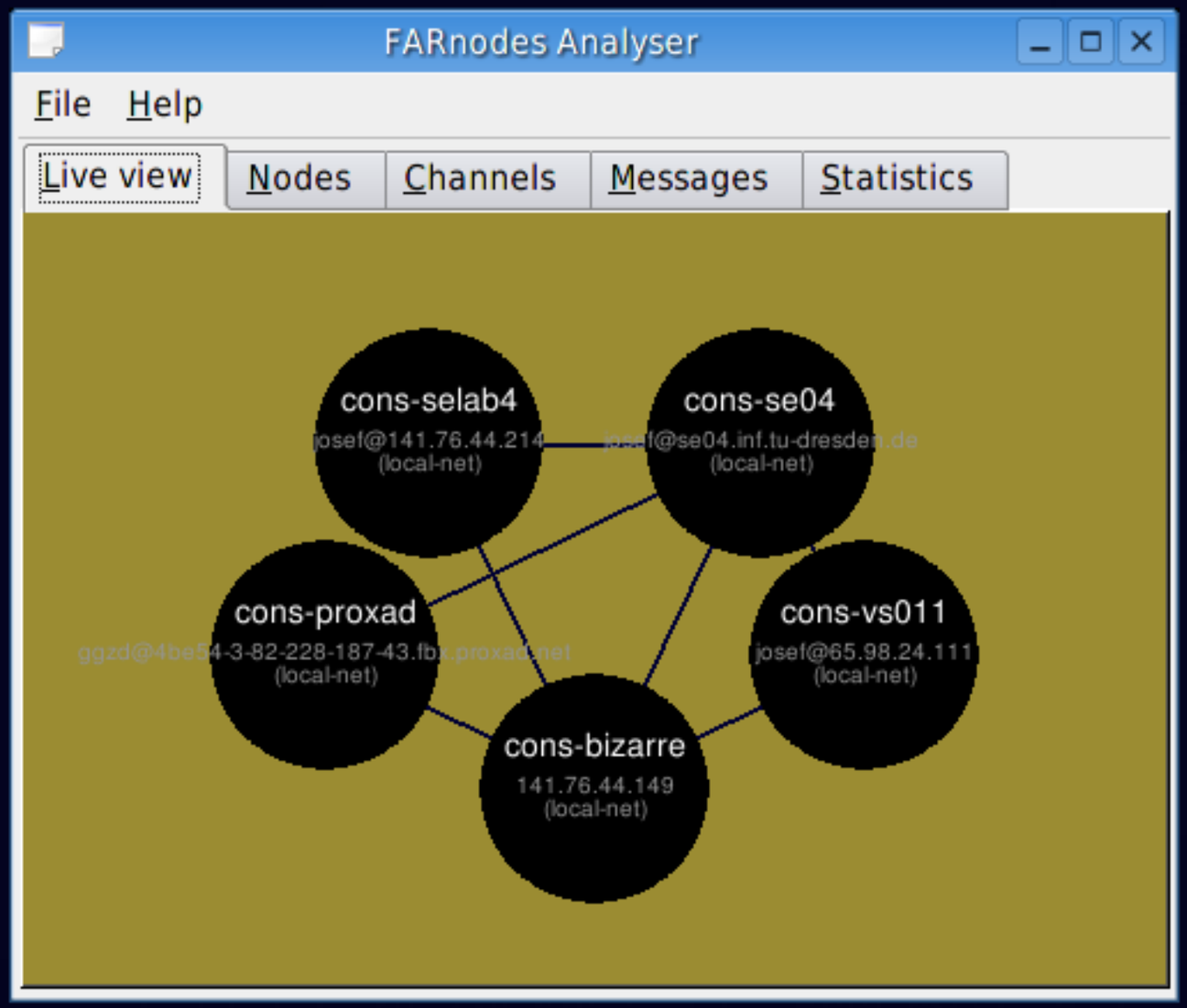
Ablaufskripte
- *.scim, reiner Python-Code
- nutzt API des FARnodes-Kernes „stubbornchannels”

Interaktion mit dem Nutzer
- Kommandozeile: sc-commandline
- grafisch: sc-simulation-topview

Werkzeuge zu FARnodes 1/2

FN Analyser

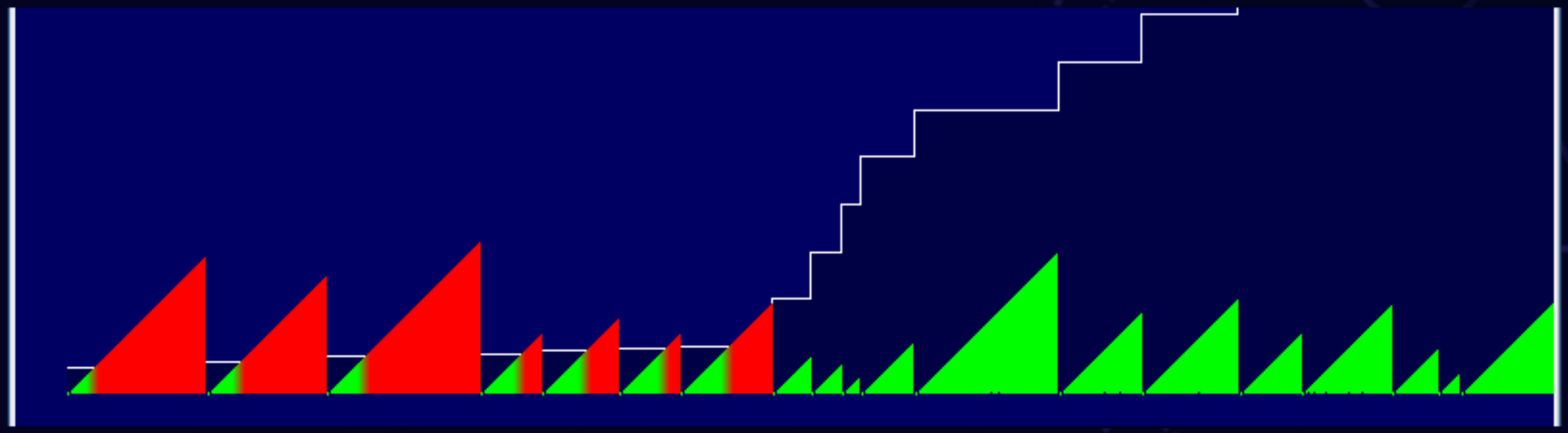
- Gewinnung der „globalen Sicht” während der Ausführung



Werkzeuge zu FARnodes 2/2

Showstats

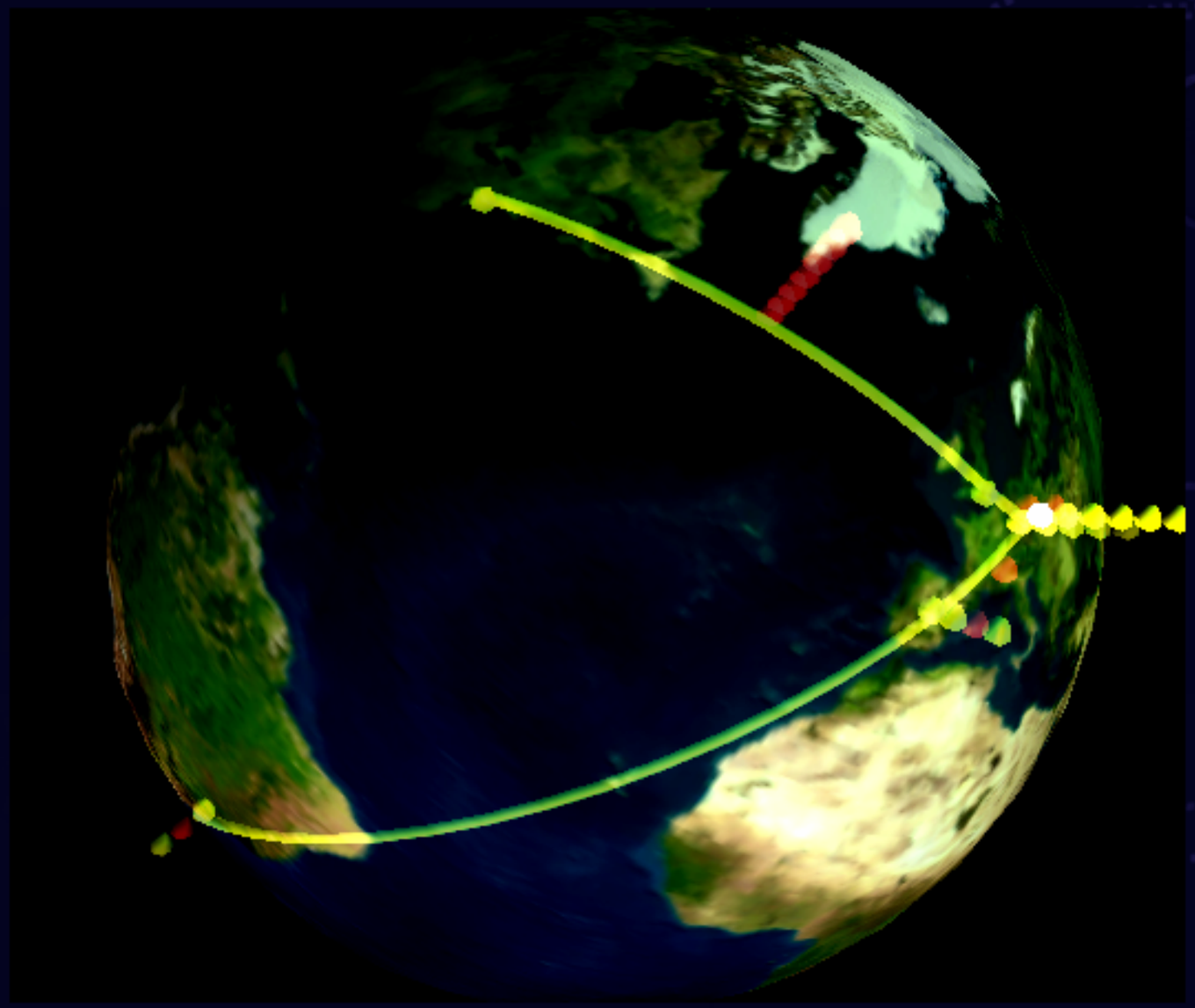
- Anzeige von EA-FD-Einstufungen
- Nachrichten werden tendentiell als schnell eingestuft



Makestats

- Extraktion von Tracedaten aus einer Logdatei
- Datenreduktion und deskriptive Statistik

Teil 4: Experimente



Experimentelle Nachweise

Lokale Simulation
- 100 virtuelle Prozesse (Nodes)
  - unter FAR-Annahmen
  - unter PS-Annahmen

Verteilte Ausführung
- im LAN der Fakultät (se01, ganymed, bizarre)
- im WAN (New Jersey, Paraná, Alsaçe)

- FN-Manager als ausführende und kontrollierende Instanz

Lokale Simulation

Simulationen sind deterministisch
- vorteilhaft für die Fehlersuche
- globale Sicht ist vorhanden

Ergebnis:
- Simulation unter FAR-Annahmen (mit EA-FD) benötigt mehr Runden
- dennoch schneller fertig aufgrund geringerer Nachrichtenanzahl

Verteilte Ausführung

Merkmale
- keine globale Sicht vorhanden
  - aber best-effort-Aggregation mit Analyser
- Startzeiten differieren erheblich, Skripte laufen unendlich
- gewisse Herausforderungen an FN-Manager
  - Proxy-Unterstützung, „Broadcast" von SIGTERM

Ergebnis:
- Consensus mit PS-FD lässt sich nicht über UDP implementieren
  - Koordinator-Deadlocks
- Anzahl der ALIVE-Nachrichten wiederum verzögernder Faktor für PS-System

Wichtig:
- Quantitative Aussagen sind von Parametern abhängig
  - wie oft versendet PS-FD eine ALIVE-Nachricht?
  - Wie hoch ist der Timeout für den Empfang?
  - Kalibrierung der Formen für EA-FD?
  - Einsatz eines vorgeschalteten Fusion-FD?

Ressourcen

FARnodes:

- http://www.inf.tu-dresden.de/~js177634/beleg/