

# Elections in a Distributed Computing System

## Proseminar Verteilte Algorithmen

Johannes Gilger

Lehrstuhl I2 RWTH Aachen

Aachen, 6. Februar 2007



*Software Modeling and Verification*

Informatik 2: Prof. Dr. Ir. Joost-Pieter Katoen

RWTHAACHEN

*moves*

- 1 Einleitung
  - Ein kleiner Überblick über Leader Election
- 2 Theorie der Leader Election
  - Voraussetzungen für unsere Algorithmen
- 3 Leader Election Algorithmen
  - Der Bully Election Algorithmus
  - Der Invitation Election Algorithmus
- 4 Praktische Anwendungen
  - IEEE 1394 - FireWire Serial Bus

## Fokus dieser Präsentation

## Das zugrundeliegende Paper

- H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE TRANS. COMP.*, 13(1):48–59, 1982.

## Begriff: Leader

*"The weaknesses of the many make the leader possible."*

**Elbert Hubbard**

### Was macht einen Leader aus?

Eine Definition, analog zum alltäglichen Gebrauchs des Wortes:

- 1 Ein Knoten in einem System von (identischen) Knoten dem eine besondere Rolle zukommt.
- 2 Ein Koordinator, eine zentrale Anlaufstelle.
- 3 Ein Knoten mit Überblick über das gesamte System.

## Begriff: Election

*“Wer die Wahl hat, hat die Qual.”*

**Sprichwort****Warum braucht man Wahlen?**

Mögliche Gründe warum man einen (neuen) Leader wählt:

- 1 Der bisherige Leader ist nicht mehr erreichbar oder verstorben.
- 2 Es gibt einen Knoten der die Aufgaben des Leaders besser erfüllen kann.
- 3 Das System hat neue Mitglieder gekriegt und man möchte sicherstellen dass diese sich integrieren, über den Leader Bescheid wissen oder vielleicht selber Leader werden.

## Arten der Wahl

### Zwei grundlegend unterschiedliche Alternativen

- 1 Wahl und Reorganisation im laufenden Betrieb.
- 2 Stop des gesamten Systems für die Wahl und Reorganisation.

## Mutual Exclusion

Leader Election ähnelt Mutual Exclusion, Koordinator zu sein einer kritischen Region.

### Unterschiede

- 1 Fairness einer Election
- 2 Koordinatoren können abstürzen
- 3 Koordinator muss andere von sich informieren

## Voraussetzungen - Teil 1

Bevor man über Leader Election sprechen kann muss man ein paar Voraussetzungen aufstellen unter denen die Wahl stattfinden soll.

### Vorraussetzungen nach Molina - Teil 1

- 1 Alle Knoten benutzen die selben Algorithmen.
- 2 Keine Bugs im Betriebssystem der Knoten.
- 3 Keine gefälschten Nachrichten oder Nachrichten deren Absender gefälscht ist.
- 4 Knoten sind mit nicht-flüchtigem Speicher ausgestattet.
- 5 Der Absturz eines Knotens verändert nichts an seinem Betriebssystem, nur Inhalte des flüchtigen Speichers gehen verloren.

## Voraussetzungen - Teil 2

### Voraussetzungen nach Molina - Teil 2

- ⑥ Es gibt keine Übertragungsfehler. Entweder eine Nachricht kommt komplett und korrekt an oder sie geht verloren.
- ⑦ Nachrichten die von einem Knoten angenommen werden, werden in der Empfangsreihenfolge abgearbeitet.
- ⑧ Das Netzwerk hat keine Störungen, es fällt *nie* aus. Kommt keine Antwort auf eine Nachricht kann man sicher sein dass der Empfänger abgestürzt ist.
- ⑨ Knoten machen keine Pause und antworten immer zeitig. Sollte dies nicht geschehen wird davon ausgegangen dass der Knoten abgestürzt ist.

## Aussagen über die Algorithmen

### Aussage 1

- 1 Wenn zwei Knoten im Status „Reorganisation“ oder „Normal“ sind müssen sie den gleichen Leader haben. Wenn beide „Normal“ sind ausserdem den gleichen Task.

Wenn bei der Wahl keine Fehler auftreten dann gilt irgendwann:

### Aussage 2

- 1 Es gibt einen Knoten im Status „Normal“ der sich für den Koordinator hält.
- 2 Alle anderen Knoten die noch laufen halten ihn auch für den Koordinator.

## Aussagen für System mit Gruppen

### Aussage 3

Die ersten beiden Aussagen müssen ebenso für ein System mit Gruppen gelten, jedoch nur innerhalb einer Gruppe und nicht über das ganze System.

## Aussagen für System mit Gruppen

## Aussage 4

Für eine maximal große Menge von Knoten  $R$ , in der alle Knoten miteinander kommunizieren können muss nach einer Wahl folgendes gelten:

- 1 Es gibt einen Knoten  $i \in R$  dessen Status „Normal“ ist und der sich selber für den Koordinator hält.
- 2 Alle anderen Knoten  $j \in R$  deren Status „Normal“ ist müssen  $i$  als Koordinator gesetzt haben und in der gleichen Gruppe wie  $i$  sein.

## Idee

Ein simpler Algorithmus zur Leader Election. Bully (eng: „to bully“ - „einschüchtern“) da der Knoten mit der höchsten id die Wahl gewinnt.

### Konzepte

- Alle Knoten haben eine einzigartige id  $1..n$ .
- Die Zahl der Knoten im System ist bekannt ( $n$ ).
- Fairness ist nicht wichtig.
- Funktioniert wenn Vorraussetzungen 1-9 gelten.

## Veranschaulichung

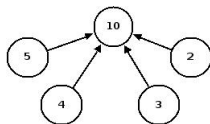
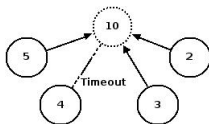
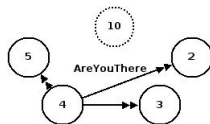
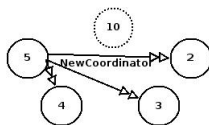
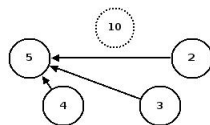
(a) Knoten 10 ist Leader  
von Knoten {2,3,4,5}(b) Knoten 10 stürzt ab,  
Knoten 4 bemerkt es(c) Knoten 4 prüft andere  
Knoten mit einer größeren id  
als es selbst.(d) Knoten 4 gibt die  
Wahl auf, Knoten 5 versucht  
gewählt zu werden(e) Da es keine Nodes mit  
 $id > 5$  gibt gewinnt Knoten  
5 die Wahl

Abbildung: Der Bully Election Algorithmus (einfaches Beispiel)

## Die Bully Election Funktion

```
1 private Election() {
2     int haltFrom, nodesTotal;
3     String status, task;
4     boolean[] up;
5
6     for (int node=id+1; node<nodesTotal; node++) {
7         if (node.AreYouThere(id) == "Yes") { exit 0; }
8     }
9     self.Stop();
10    status = "Election";
11    haltFrom = id;
12    for (int node=id-1; node>0; node--) {
13        if (node.Halt(id) == "Ok") { up[node] = true; }
14    }
15    coordinator = id;
16    status = "Reorganization";
17    for (int node=0; node<= up.length; node++) {
18        if (up[node] == true)
19            if (node.NewCoordinator() != "Ok") {
20                self.Election(); exit 0; }
21            if (node.Ready(task) != "Ok") {
22                self.Election(); exit 0; }
23        }
24    }
25    status = "Normal";
26 }
```

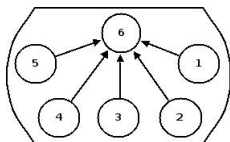
## Idee

Der Bully war ja schon gut, aber was ist wenn doch mal das Kommunikationsnetzwerk (teilweise) ausfällt?

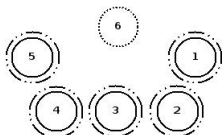
### Konzepte

- Es gibt mehrere Gruppen/Partitionen.
- Jede Gruppe muss einen eindeutigen Leader haben der in dieser Gruppe ist.
- Jede Gruppe sollte größtmöglich sein.
- Koordinatoren schicken Einladungen an anderen Gruppen (Invitation).
- Funktioniert wenn Voraussetzungen 1-7 gelten.

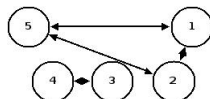
## Veranschaulichung



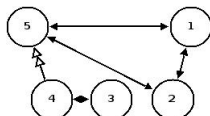
(a) Node 6 ist Leader von Nodes {1,2,3,4,5}



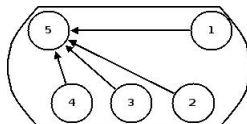
(b) Node 6 stürzt ab, Nodes 1-5 formen eigene Gruppen



(c) Nodes 4 und 5 schicken gleichzeitig Invitations. Node 3 nimmt Einladung von 4 an, Nodes 1 und 2 die von 5



(d) Etwas später lädt ein Leader den anderen ein



(e) Die Gruppen haben sich vereinigt

Abbildung: Der Invitation-Algorithmus (Beispiel wie in [1])

## Funktionen und Variablen im Invitation Algorithmus

```
1 public Check() {}
2 // Regelmässig vom Betriebssystem aufgerufen
3 private Merge(boolean[] coordinators) {}
4 // Wird bei Bedarf von Check() aufgerufen
5 public TimeOut() {}
6 // Wird regelmässig aufgerufen
7 public Ready(int readyFrom, int groupid, String task) {}
8 // Wird nach einer Wahl vom neuen Coordinator aufgerufen
9 public AreYouCoordinator() {}
10 // Wird bei einer Wahl vom Koordinator aufgerufen
11 public AreYouThere(int groupid) {}
12 // "Ping-Befehl"
13 public Invitation(int groupid) {}
14 // Wird von anderen Coordinatoren aufgerufen
15 public Accept(int groupid) {}
16 // Einladung wurde akzeptiert
17 private Recovery() {}
18 // Nach einem Crash, beim Hochfahren
```

```
1 String status, task;
2 int id, groupid, counter, node, seconds, setMax;
3 boolean[] neighbours, coordinators;
```

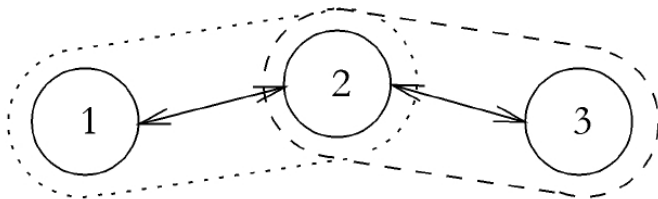
## Invitation-Check Funktion

```
1 private Check() {
2   if (status == "Normal" && coordinator == id) {
3     for (node=1; node<nodesTotal; node++) {
4       if (node.AreYouCoordinator() == "Yes") {
5         coordinators[node] = true; }
6     }
7     for (int i=n; i>=1; i--) {
8       if (coordinators[i] == true) {
9         setMax = i; } }
10    if (id < setMax) { sleep setMax-id; }
11    self.Merge(coordinators);
12  }
13 }
```

## Die Invitation Merge Funktion

```
1 private Merge (boolean[] coordinators) {
2   status = "Election";
3   self.Stop();
4   counter++;
5   groupid = id . counter;
6   coordinator = id;
7   for (node=1; node<nodesTotal; node++) {
8     if (coordinators[node]) {
9       node.Invitation(id, groupid);
10    else if (neighbours[node]) {
11      node.Invitation(id, groupid); }
12  }
13  neighbours = [];
14  sleep(seconds);
15  status = "Reorganization";
16  /* Here the reorganization takes place.
17   * After that the "neighbours" array
18   * contains the nodes which
19   * accepted the invitation. */
20  for (node=1; node<neighbours.length; node++) {
21    if (neighbours[node]) {
22      if (node.Ready(id, groupid, task) != "Ok") {
23        self.Recovery(); } } }
24  status = "Normal"
25 }
```

## Probleme



Widerspruch zur Aussage 4 nach Stoller ([7]).

Gruppe 1:  $\{Node1, Node2\}$ , Leader:  $Node1$ .

Gruppe 2:  $\{Node2, Node3\}$ , Leader: Sollte für beide Nodes  $\in \{2, 3\}$  sein, ist aber nicht möglich  $\rightarrow$  Widerspruch!

## FireWire: Leader Election

### Leader Election bei FireWire - Begriffe

**Bus** Der FireWire Bus, also der Bus am Computer z.B.

**Node** Gerät was an den Bus (direkt/indirekt) angeschlossen wird.

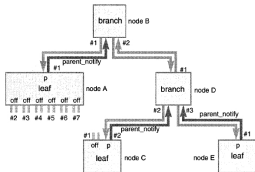
**Port** FireWire Ports an den Nodes.

**Connection** Verbindung eines Ports mit einem anderen (maximal eine).

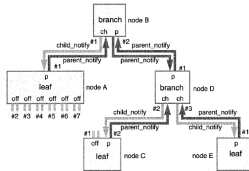
**Leaf** Node mit nur einer Verbindung.

**Branch** Node mit mehr als einer Verbindung.

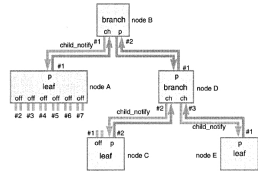
# FireWire: Leader Election & Root Contention



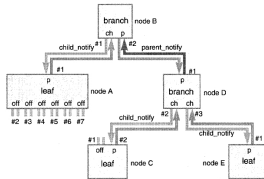
(a) Parent Notify



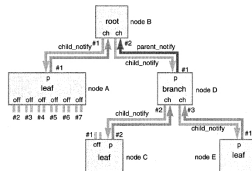
(b) Child Notify



(c) Root Contention Scenario



(d) Root Contention



(e) Root Contention resolved

# The End

Vielen Dank für's Zuhören



H. Garcia-Molina.

Elections in a Distributed Computing System.

*IEEE TRANS. COMP.*, 13(1):48–59, 1982.



IEEE.

*IEEE 1394 Standard for a High Performance Serial Bus*, 1995.

<http://standards.ieee.org/catalog/olis/busarch.html>.



MF Kaashoek and AS Tanenbaum.

Group communication in the Amoeba distributed operating system.

*Distributed Computing Systems, 1991., 11th International Conference on*, pages 222–230, 1991.



L. Lamport, R. Shostak, and M. Pease.

The Byzantine Generals Problem.

*ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.



N. Malpani, J.L. Welch, and N. Vaidya.

Leader election algorithms for mobile ad hoc networks.

*Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, 2000.



M. Stoelinga and F. Vaandrager.

Root contention in IEEE 1394.

*Lecture notes in computer science*, pages 53–74.



S.D. Stoller.

Leader Election in Asynchronous Distributed Systems.

*IEEE Transactions on Computers*, 49(3):283–284, 2000.



K. Weniger and M. Zitterbart.

IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks.