Leader election con CSP

Nota Preliminare

- La presente dispensa è una rivisitazione di
  - Yucheng Fang, Huibiao Zhu, Huiwen Wang, *Modeling and Verifying Leader Election Algorithm in CSP*, Proc. of the 30th Int.l Conf. on Software Engineering and Knowledge Engineering, July, 1-3, 2018 S. Francisco Bay – USA, pp.342-347
  - (disponibile all'indirizzo http://ksiresearchorg.ipage.com/seke/Proceedings/seke/SEKE2018_Proceedings.pdf)

Generalità (1/2)

- In una rete per leader election si intende l'algoritmo che permette di identificare un nodo considerate come leader dagli altri
- Scopo
  - Permettere il controllo della rete da parte del leader al fine di svolgere alcune operazioni con gli altri host

Generalità (2/2)

- Esempi di operazioni che richiedono un leader: distribuzione di chiavi, coordinamento nel routing, controlli di natura generale
- Esistono vari algoritmi per la LE nelle reti wired, ma pochi per la MANET
- In generale, LE identifica il nodo con massimo valore rispetto ad alcuni criteri (es. Tempo di vita della batteria)

LE in Sintesi (1/3)

- Quando ha inizio l'elezione del leader, da parte di un nodo (sorgente dell'elezione), questi invia in broadcast ai suoi vicini un messaggio election
- Ogni nodo che riceve election per la prima volta registra il mittente come suo parent in un albero e reinvia il msg election

LE in Sintesi (2/3)

- Quando un nodo riceve un msg election promosso da un nodo che non è suo parent, risponde con un ack
  - Ack contiene, oltre ad alter informazioni, anche il valore della misura rispetto alla quale viene scelto il leader
- Quando un nodo ha ricevuto gli ack da tutti i suoi figli, invia ack al proprio parent

LE in Sintesi (3/3)

- Quando la sorgente ha ricevuto tutti gli ack conosce i valori della misura per tutti i nodi della rete, quindi:
  - Scegli il leader
  - Invia in broadcast il msg leader
- Quando un nodo riceve il msg leader aggiorna la propria lista di leader e reinvia in broadcast lo stesso messaggio

Elezioni Multiple

- Più nodi possono concorrentemente dare inizio a un'elezione
- In tal caso una sola sopravvive:
  - a ogni elezione è associate una priorità
  - ogni nodo già coinvolto in un'elezione ignora le elezioni successive con priorità inferiore
  - partecipa però alle elezioni successive con priorità superiore

Fallimento

- Talvolta un nodo potrebbe NON inviare un ack, per es. a causa del movimento dei nodi
- Ogni nodo attende ack per un intervallo di tempo predefinito, trascorso il quale il nodo che non ha inviato ack viene rimosso

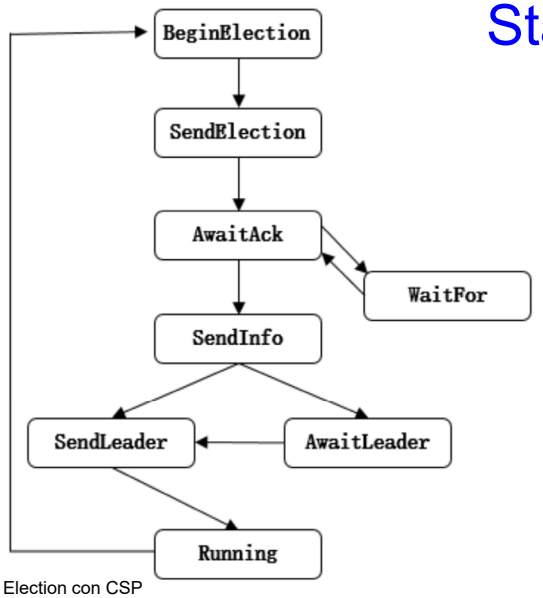
Modellazione: Parametri del Modello (1/2)

- Messaggi possibili:
  - Election
  - Ack
  - Leader
  - Probe (usato per determinare se un nodo è ancora attivo)
  - Reply (inviato in risposta a un probe)

Modellazione: Parametri del Modello (2/2)

- Variabili usate da ogni processo per decidere l'azione da svolgere:
  - d_i: indica se il nodo i è coinvolto in una elezione
  - p_i: nodo parent del nodo i
  - D_i: indica se il nodo i ha spedito ack a p_i
  - lid_i: leader del nodo i
  - N_i: insieme di nodi vicini di i
  - S_i: insieme di nodi da cui i non ha ricevuto ack
  - src_i: priorità del nodo i
  - max_i: nodo con valore massimo di cui i è a conoscenza

Stati



BeginElection

SendElection(n,True,n,false,-1,N,N,n,n,N)

SendElection

$$
\begin{aligned}
&SendElection(n,d,p,D,lid,N,S,src,max,N') \mathrel{\widehat{=}} \\
&\quad if\ empty(N')\ then\ AwaitAck(n,d,p,D,lid,N,S,src,max) \\
&\quad else\ election.n!i : N'!src \rightarrow SendElection(n,d,p,D,lid,N,S,src,max,N' \setminus \{i\}) \\
&\quad \Box election?c : N!n?s \rightarrow (if\ s.p > src.p\ then\ SendElection(n,d,c,False,-1,N,N \setminus \{c\},s,max,N \setminus \{c\})) \\
&\qquad\qquad else\ ack!n!c!max \rightarrow SendElection(n,d,p,D,lid,N,S,src,max,N')) \\
&\quad \Box ack?c : S!n?v \rightarrow SendElection(n,d,p,D,lid,N,S \setminus \{c\},src,Max(max,v),N') \\
&\quad \Box leader?c : N!n?s \rightarrow (if\ s.p > src.p\ then\ SendElection(n,d,c,False,-1,N,N \setminus \{c\},s,max,N \setminus \{c\}) \\
&\qquad\qquad else\ SendElection(n,d,p,D,lid,N,S,src,max,N')) \\
&\quad \Box probe?c \in N!n \rightarrow reply!c!n \rightarrow SendElection(n,d,p,D,lid,N,S,src,max,N') \\
&\quad \Box fail.n \rightarrow Faild(n,N) \\
&\quad \Box tock \rightarrow SendElection(n,d,p,D,lid,N,S,src,max,N')
\end{aligned}
$$

AwaitAck

$$
\begin{aligned}
&AwaitAck(n,d,p,D,lid,N,S,src,max) \mathrel{\widehat{=}} \\
&\quad if\ empty(S)\ then\ SendInfo(n,d,p,D,lid,N,S,src,max) \\
&\quad else\ tock \rightarrow AwaitAck(n,d,p,D,lid,N,S,src,max) \\
&\quad \Box ack?c : S!n?v \rightarrow AwaitAck(n,d,p,D,lid,N,S \setminus \{c\},src,Max(max,v)) \\
&\quad \Box probe?c : N!n \rightarrow reply!c!n \rightarrow AwaitAck(n,d,p,D,lid,N,S,src,max) \\
&\quad \Box probe!n?j : S \rightarrow WaitFor(n,d,p,D,lid,N,S,src,max,j,T) \\
&\quad \Box tock \rightarrow AwaitAck(n,d,p,D,lid,N,S,src,max) \\
&\quad \Box fail.n \rightarrow Faild(n,N) \\
&\quad \Box election?c : N!n?s \rightarrow (ifs.p > src.p \\
&\qquad then\ SendElection(n,d,c,False,-1,N,N \setminus \{c\},s,max,N \setminus \{c\}) \\
&\qquad else(if\ s == src\ then\ ack!n!c!max \rightarrow AwaitAck(n,d,p,D,lid,N,N \setminus \{c\},src,max) \\
&\qquad\quad else\ AwaitAck(n,d,p,D,lid,N,S,src,max))) \\
&\quad \Box leader?c : N!n?s \rightarrow (if\ s.p > src.p \\
&\qquad then\ SendElection(n,d,c,False,-1,N,N \setminus \{c\},s,max,N \setminus \{c\}) \\
&\qquad else\ election.n!c!src \rightarrow AwaitAck(n,d,p,D,lid,N,S,src,max))
\end{aligned}
$$

WaitFor

if T == 0

then AwaitAck(n, d, p, D, lid, N, S, \{pid}, src, max)

else tock → WaitFor(n, d, p, D, lid, N, S, src, max, pid, T-1)

□ reply.n.pid → AwaitAck(n, d, p, D, lid, N, S, src, max)

SendInfo

```
if src == n
then SendLeader(n,d,p,D,max,N,S,src,max,N)
else ack!n!p!max →
    AwaitLeader(n,d,p,True,max,N,S,src,max)
□ tock → SendInfo(n,d,p,D,lid,N,S,src,max)
□ fail.n → Faild(n,N)
□ probe.p.n → reply.p.n →
    SendInfo(n,d,p,D,lid,N,S,src,max)
□ election?c : N!n?s → (if s.p > src.p then
        SendElection(n,d,c,False,-1,N,N\{c},s,max,N\{c})
else SendInfo(n,d,p,D,lid,N,S,src,max))
```


Faild

```
tock → Faild'(n,N)
□ probe?c : N!n → Faild(n,N)
□ election?c : N!n?v → Faild(n,N)
□ ack?c : N!n?v → Faild(n,N)
□ leader?c : N!n?s → Faild(n,N)
```


Faild'

```
tock → Faild'(n,N)
□ probe?c : N!n → Faild(n,N)
□ election?c : N!n?v → Faild(n,N)
□ ack?c : N!n?v → Faild(n,N)
□ leader?c : N!n?s → Faild(n,N)
□ revive.n → BeginElection(n,N)
```


SendLeader

```
if empty(N')
    then Running(n,false,n,D,lid,N,S,src,max)
    else leader.n!i : N'!max →
        SendLeader(n,d,p,D,lid,N,S,src,max,N'\{i})
□ tock → SendLeader(n,d,p,D,lid,N,S,src,max,N')
□ fail.n → Faild(n,N)
□ election?c : N!n?s →
    if s.p > src.p then
        SendElection(n,d,c,False,-1,N,N\{c},s,max,N\{c})
    else SendLeader(n,d,p,D,lid,N,S,src,max,N')
```

AwaitLeader

Leader.p.n?v →SendLeader(n,d,p,D,v,N,S,src,v,N\{p})

□ tock → AwaitLeader(n,d,p,True.lid,N,S,src,max)

□ fail.n → Faild(n,N)

□ probe.p.n → reply.p.n →
AwaitLeader(n,d,p,D,lid,N,S,src,max)

□ election?c : N!n?s → if s.p > src.p then
SendElection(n,d,c,False,-1,N,N\{c},s,max,N\{c})
else if s == src then ack!n!c!max !→
AwaitLeader(n,d,p,D,lid,N,S,src,max)
else AwaitLeader(n,d,p,D,lid,N,S,src,max)

Running

□ election?c : N!n?v → (p(v) > p(lid)or ¬d)&
SendElection(n,d,c,False,-1,N,N\{c},v,max,N\{c,v}))

□ tock → Running(n,d,p,D,lid,N,S,src,max)

□ fail.n → Faild(n,N)

□ probe?c : N!n → reply!c!n →
Running(n,d,p,D,lid,N,S,src,max)

Analisi

- È possible dimostrare che:
  - Il Sistema non presenta rischi di deadlock
  - L'algoritmo individua un unico leader