Leader Election Problem

Corso di laurea magistrale in Informatica
Corso di Algoritmi avanzati
2010-2011
Docente: Rossella Petreschi
Relatore: Alessandro Gaeta

Università La Sapienza (Roma)

Introduzione

- **Problema:**

  insieme di processori collegati e comunicanti mediante messaggi, ***scelta del leader***.

- **Topologia:**

  Anello

  Struttura conveniente perchè corrisponde a sistemi di comunicazione fisici reali come ad esempio i token ring

Introduzione

- Rilevanza della Leader Election:
  - Problemi di symmetry breaking.

    Risolvere una situazione di deadlock tramite l'elezione di un leader.

  - Fault tollerance e preservazione delle risorse.

    Un processore leader offre una semplice soluzione per il problema del broadcast

Definizione

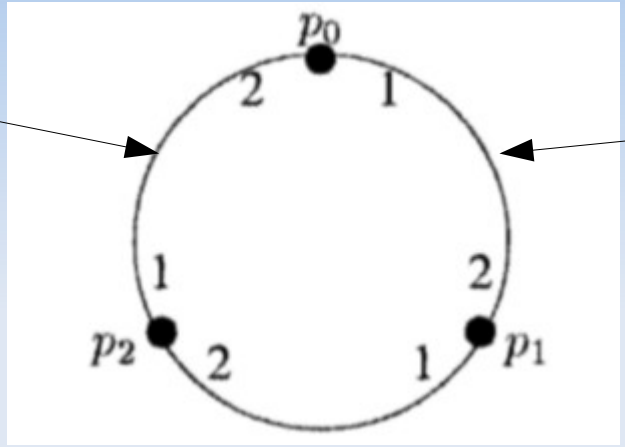
- Informale:
  - Ogni processore all'interno della rete deve decidere se essere o meno il leader.
  - Solo uno dei processori può essere eletto.
- Formale:

  Un algoritmo è in grado di risolvere il problema se soddisfa le seguenti condizioni:

  - Esistono solo due stati **eletto** e **non eletto**.
  - In ogni possibile esecuzione entra nello stato eletto un solo processore mentre tutti gli altri restano nello stato non eletto

Esempio

Arco destro del processore p_0



Arco sinistro del processore p_0

Trattando reti ad anello, ogni processore avrà un **identificativo univoco p_i**, mentre gli archi sono etichettati secondo le seguenti regole:

Un arco che va **da p_i a p_{i+1}** è etichettato con **valore 1**
(arco sinistro o arco in senso orario)

Un arco che va **da p_i a p_{i-1}** è etichettato con **valore 2**
(arco destro o arco in senso antiorario)

Anello anonimo

- **Definizione:**

  Un sistema si dice **anonimo** se i processori non hanno un identificativo univoco.

- In questo caso l'anello può essere descritto solo nei termini dell'etichettatura degli archi mentre tutti i processori si troveranno nello stesso stato.

Teorema: *Non esiste un algoritmo anonimo per il problema del leader election.*

Anello anonimo

Nota:
Si proverà il risultato per una rete sincrona con algoritmo non-uniforme.

Dimostrazione informale:

In un sistema sincrono, ogni algoritmo procede per round in ciascuno dei quali, tutti i messaggi vengono spediti. L'idea alla base della dimostrazione è quindi, che la simmetria del sistema può essere sempre mantenuta

Anello anonimo

- **Algoritmo:**

1) Tutti i processori partono da uno stesso stato iniziale;
2) Dato che i processori sono identici ed eseguono lo stesso programma, mandano tutti esattamente lo stesso messaggio;
3) Tutti i processori ricevono lo stesso messaggio ad ogni round;
4) Se un processore viene eletto come leader allora tutti i processori sono eletti a leader.

Anello anonimo

Dimostrazione formale:

Lemma:

Per ogni round k di qualunque possibile esecuzione di un algoritmo anonimo su un anello R lo stato di tutti i processori è lo stesso alla fine del round k.

Dimostrazione:

Passo base **k=0** (prima del primo round) è semplice in quanto tutti i processori partono dallo stesso stato iniziale

Anello anonimo

Passo induttivo: Si assuma che il lemma valga per il round k-1. Siccome i processori sono tutti nello stesso stato al round k-1 allora essi inviano lo stesso messaggio m_r (sull'arco destro) e lo stesso messaggio m_l (sull'arco sinistro). Di conseguenza al round k tutti i processori ricevono lo stesso messaggio sui rispettivi archi e siccome eseguono lo stesso programma, si troveranno nello stesso stato alla fine del round k.

Alla fine di ogni round i processori si trovano sempre nello stesso stato, quindi se un processore dichiara se stesso leader allora tutti i processori dichiareranno loro stessi come leader.

Asynchronous Ring

Si considerino **reti asincrone** in cui ogni processore ha un identificativo univoco (ad esempio un numero naturale). Così si **può specificare un anello mediante l'elencazione degli identificativi in ordine orario a partire da quello di valore minore**.

Presenteremo di seguito un upper ed un lower bound per la complessità, in termini di messaggi inviati da un algoritmo che risolve il problema del leader election in un anello.

Upper bound $O(n^2)$

Ad ogni processore pi è assegnato un identificativo id_i con $0 \leq i < n$.

- **Algoritmo:**
  1) In ogni round ogni processore invia un messaggio contenente il suo identificativo al suo vicino sinistro e attende un messaggio dal suo vicino destro
  2) Quando un processore riceve un messaggio controlla l'identificativo che vi è contenuto:
  3) Se l'identificativo ricevuto è maggiore del proprio, allora reinvia il messaggio al suo vicino sinistro

Upper bound $O(n^2)$

4) Se l'identificativo contenuto nel messaggio è minore del proprio allora blocca il messaggio e non lo reinvia al proprio vicino.

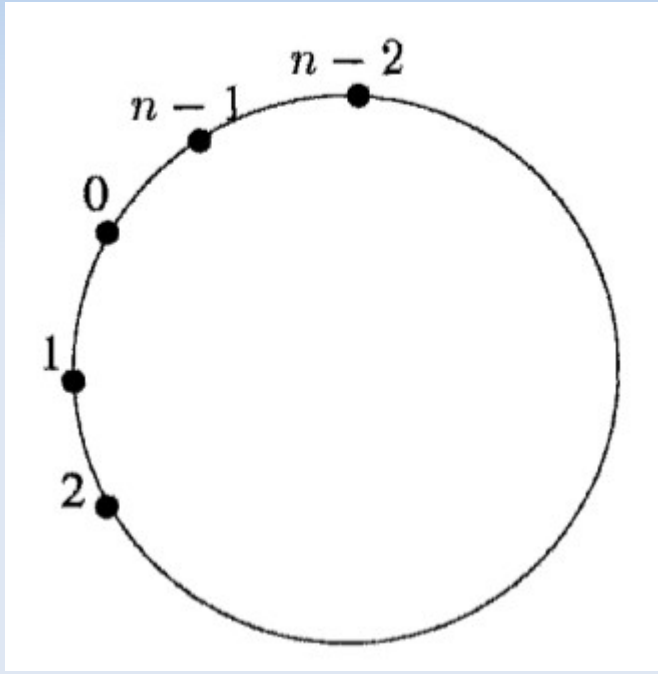
5) Se riceve un messaggio che contiene il proprio identificativo allora si auto-dichiara leader, invia un messaggio di terminazione al suo vicino sinistro e termina la computazione come leader.

6) Se riceve un messaggio di terminazione lo reinvia al suo vicino e termina nello stato non eletto.

Correttezza: Solo il messaggio con l'identificativo maggiore non viene mai bloccato; dunque solo il processore con l'identificativo maggiore dichiarerà se stesso come leader.

Upper bound $O(n^2)$

L'algoritmo non invia mai più di $O(n^2)$ messaggi.
Nel caso peggiore in cui i processori sono ordinati in ordine decrescente in base al valore del loro identificativo (come in figura), il messaggio del generico processore i è inviato esattamente i+1 volte.
Di conseguenza il numero totale di messaggi (compresi gli n messaggi di terminazione) inviati è:



$$n + \sum_{i=0}^{n-1}(i+1) = \Theta(n^2)$$

Un Algoritmo O(n log n)

Definizione k-neighborhood(p_i):

insieme dei processori che si trovano al più a distanza k da p_i .

La k-neighborhood di un processore include esattamente 2k+1 processori

Fasi:

- Nella k-esima fase un processore tenta di diventare il vincitore della fase. Per esserlo dovrà avere l'identificativo più alto all'interno del suo 2^k-neighborhood.
- Solo i processori vincitori nella k-esima fase possono partecipare alla fase k+1

Un Algoritmo O(n log n)

- In particolare
  - Fase 0: ogni processore tenta di diventare vincitore inviando un ”messaggio sonda” al suo 1-vicinato. Ogni processore che riceve un ”messaggio sonda” blocca il messaggio se il suo identificativo è maggiore rispetto a quello contenuto nel messaggio; altrimenti invia indietro un ”messaggio risposta”. Se un processore riceve un ”messaggio risposta” da entrambi i suoi vicini allora è il **vincitore della fase**.
  - Fase k: il generico processore vincitore nella fase k-1 manda il ”messaggio sonda” nel suo 2^k-vicinato, tale messaggio attraversa i 2^k processori.

Un Algoritmo O(n log n)

La sonda verrà bloccata se uno dei processori che la riceve ha identificativo maggiore.

- Se la sonda arriva all'ultimo processore allora questo invia un ”messaggio di risposta”. Se un processore riceve un messaggio risposta su entrambi i suoi archi diventa vincitore della fase k ed accede alla fase successiva.
- Se un processore riceve il suo messaggio sonda allora si auto-dichiara leader e termina la computazione inviando un messaggio di terminazione

Correttezza: Le politiche di blocco di un messaggio simili a quelle del algoritmo precedente, assicurano la correttezza.

Un Algoritmo O(n log n)

Analisi al Caso peggiore.

- **Oss.1** Nella fase k ogni sonda percorre 2^k processori ed il numero di messaggi inviati è $4*2^k$
- **Oss.2** Per ogni k≥1 il numero di processori che partecipano alla fase k+1 è al più $\frac{n}{2^k+1}$

  Se un processore p_i è vincitore nella fase k allora i processori nel suo 2^k-vicinato devono avere identificativo minore.

  Sia p_j un altro processore, affinchè anch'esso sia vincitore nella fase k, p_i e p_j devono essere almeno a distanza 2^k,ovvero ci sono almeno 2^k processori tra p_i e p_j. Nel caso limite in cui tutti i vincitori sono esattamente a distanza 2^k tra loro otteniamo dunque che il numero di vincitori possibili è $\frac{n}{2^k+1}$

Un Algoritmo O(n log n)

Dal lemma precedente deriviamo che **vi è un solo vincitore se la fase è almeno log(n-1).**

Di conseguenza considerando che ogni messaggio è inviato $4*2^k$ volte, che il numero di vincitori ad ogni fase è al più $n/2^k+1$ e considerati i 4n messaggi della fase iniziale e gli n della fase di terminazione otteniamo che il numero di messaggi inviati è al più:

$$5n + \sum_{k=1}^{\lceil \log(n-1) \rceil + 1} 4 \cdot 2^k \cdot \frac{n}{2^{k-1}+1} < 8n(\log n + 2) + 5n$$

Dunque O(n log n) C.V.D.

Lower Bound $\Omega(n \log n)$

Teorema: *Qualsiasi algoritmo per il problema del leader election in un anello asincrono richiede l'invio di $\Omega(n \log n)$ messaggi.*

Proveremo tale lower bound per una versione leggermente modificata dell'algoritmo in cui:

- Il leader sarà il processore con l'identificativo maggiore.
- Tutti i processori conoscono l'identificativo del leader eletto

Lower Bound $\Omega(n \log n)$

Idea: Si costruirà un algoritmo dispendioso per un anello di dimensione n/2. Successivamente si uniranno due anelli di dimensione n/2, in uno di dimensione n combinando l'esecuzione dei due algoritmi.

Nel dimostrare tale lower bound si farà uso di **scheduler**, in modo da applicare la stessa sequenza di eventi in anelli diversi e con diversi identificativi per i processori.

Definizione: uno scheduler σ di un algoritmo A per un anello è detto **aperto** se esiste un arco ***e*** tale che in σ nessun messaggio è inviato attraverso ***e*** in entrambe le direzioni. Cosi definito ***e*** è un arco aperto di σ

Lower Bound Ω(n log n)

Dato che l'algoritmo è uniforme, i processori non conoscono la grandezza dell'anello.È così possibile unire due scheduler di due anelli più piccoli, per crearne uno di un anello più grande. Per semplicità si assuma che n sia una potenza di 2.

Teorema: *Per ogni n e ogni insieme n di identificativi esiste un anello che usa tali identificativi come un open scheduler di A, nel quale almeno M(n) messaggi sono ricevuti. In tal caso M(2)=1 ed* $M(n)=2\mathrm{M}(\frac{n}{2})+\frac{1}{2}(\frac{n}{2}-1)$

Lower Bound $\Omega(n \log n)$

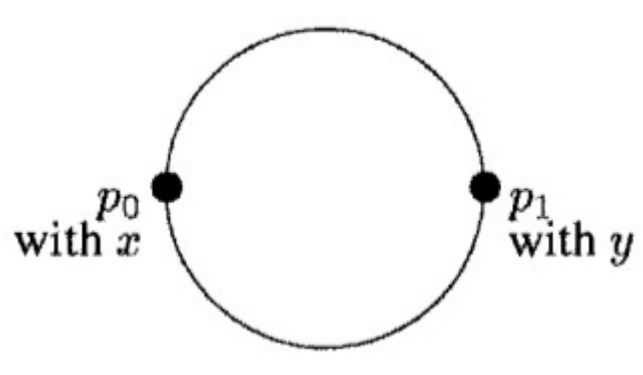
- Caso base n=2.

  Per ogni insieme di due identificativi, esiste un anello R che usa i due identificativi in modo tale che siano un open scheduler di A in cui almeno un messaggio è ricevuto.

  Dim: Assumiamo che R contenga i due processori p_0(con identificativo x) e p_1(con identificativo y). Sia A un algoritmo per R e sia δ un ammissibile esecuzione di A, allora affinché A sia corretto almeno un messaggio deve essere ricevuto in δ altrimenti p_1 non può conoscere l'identificativo di p_0.

Lower Bound $\Omega(n \log n)$

Sia σ il più corto prefisso di δ che includa il primo evento di ricezione di messaggio. Si noti allora che, l'arco successivo a quello in cui è stato ricevuto il primo messaggio è un arco aperto. In tal modo, esattamente un solo messaggio è stato ricevuto in σ ed un arco è aperto. σ è dunque un open scheduler.



Lower Bound $\Omega(n \log n)$

- Passo induttivo.
  - Si prendano due open-scheduler in due anelli di dimensione minore per concatenarli in corrispondenza degli archi aperti in modo da ottenere un solo scheduler per un anello più grande in cui gli stessi messaggi (più altri extra) sono ricevuti.
  - Una volta uniti gli scheduler, i processori nel mezzo hanno necessità di conoscere l'identificativo del leader e ciò comporta lo scambio dei messaggi ulteriori.

Lower Bound Ω(n log n)

- *<u>Lemma:</u> Si assuma che per ogni set di n/2 identificativi, esiste un anello che usa tali identificativi come un open scheduler di A nel quale M(n/2) messaggi sono ricevuti. Allora per ogni set di n identificativi, esiste un anello che usa tali identificativi come un open schedule in cui* $M(n)=2\mathrm{M}(\frac{n}{2})+\frac{1}{2}(\frac{n}{2}-1)$ *messaggi sono ricevuti.*

<u>Dimostrazione:</u>
Sia S un insieme di *n* identificativi.
Si partizioni S in due sottoinsiemi di grandezza n/2
Esistono allora due anelli R_1 ed R_2 che hanno due open scheduler σ_1 ed σ_2 in cui al meno M(n/2) messaggi sono ricevuti.

Lower Bound $\Omega(n \log n)$

Siano e_1 de e_2 i rispettivi archi aperti e (p_1,p_2) ed (q_1,q_2) gli estremi di tali archi.

Colleghiamo R_1 ad R_2 tramite l'eliminazione di e_1 ed e_2 e l'inserimento degli archi e_p ed e_q (come in figura).

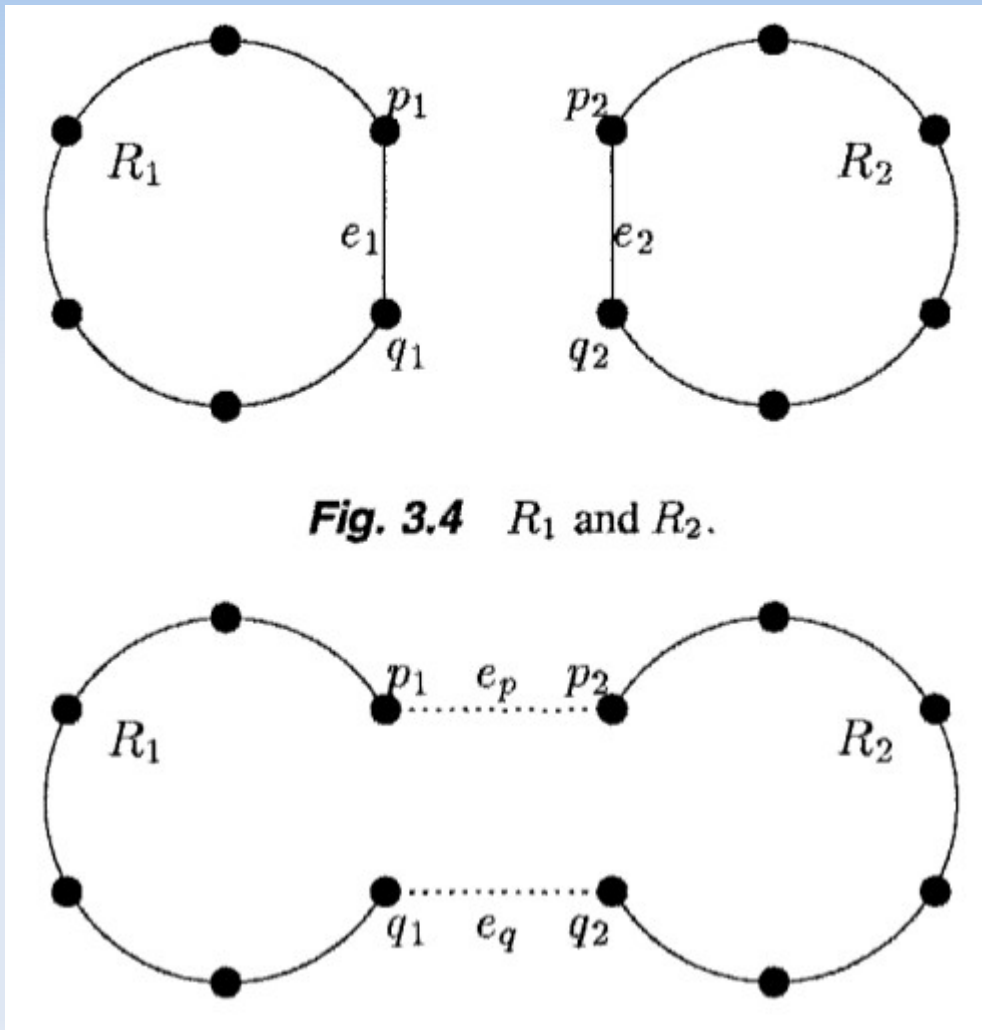


Fig. 3.4 R_1 and R_2.

Lower Bound $\Omega(n \log n)$

- I processori in R_1 (o in R_2) non possono distinguere se R_1 (o R_2) è un anello indipendente o un sotto-anello di R. Cosi $\sigma_1(\sigma_2)$ viene visto come se fosse indipendente.
- Di conseguenza concatenando σ_1 σ_2 otteniamo un open scheduler per R in cui almeno 2M(n/2) messaggi sono ricevuti.

  Dimostriamo ora che che è necessario ricevere 1/2(n/2 – 1) ulteriori messaggi.

  Consideriamo uno scheduler σ_1 σ_2 σ_3 di R, in cui sia e_p che e_q sono archi aperti. Se esiste uno schedule in cui 1/2 (n/2 – 1) messaggi sono ricevuti in σ_3 il lemma è provato.

Lower Bound $\Omega(n \log n)$

- Supponiamo che esista uno **scheduler** σ1 σ2 σ3 ***inattivo*** rispetto all'esecuzione
- Un **processore è inattivo** se non invia alcun messaggio fintanto che non ne riceva uno
- Una **configurazione è inattiva** rispetto ad e_p ed e_q se nessun messaggio è in transito se non sugli archi e_p , e_q (tutti gli altri processori sono inattivi).
- Supponendo ora che il processore con il massimo identificativo si trovi in R_1, fintanto che nessun messaggio è spedito da R_1 ad R_2 i processori in R_2 non possono terminare. Di conseguenza ogni processore in R_2 deve ricevere almeno un altro messaggio per terminare. Si richiede quindi un ulteriore invio di Ω(n/2) messaggi.

Lower Bound $\Omega(n \log n)$

Lemma: Esiste un segmento di scheduler finito σ_4 in cui $1/2(n/2 - 1)$ messaggi sono ricevuti, tale che $\sigma_1\sigma_2\sigma_3\sigma_4$ è un open scheduler in cui solo uno tra e_p ed e_q è aperto.

Dimostrazione: Sia σ_4'' tale che $\sigma_1\sigma_2\sigma_3\sigma_4''$ sia uno scheduler ammissibile, così che tutti i messaggi sono inviati tramite e_p ed e_q e tutti i processori terminano. Almeno n/2 messaggi sono ricevuti in σ_4''. Sia σ_4' il più corto prefisso di σ_4'' in cui sono ricevuti n/2 - 1 messaggi. σ_4' inizia in una configurazione inattiva in cui i messaggi transitano solo su e_p ed e_q. Costruiamo dunque due insiemi P e Q, dove P contiene i processori risvegliati del msg su e_p e contiene uno solo tra p_1 e p_2.

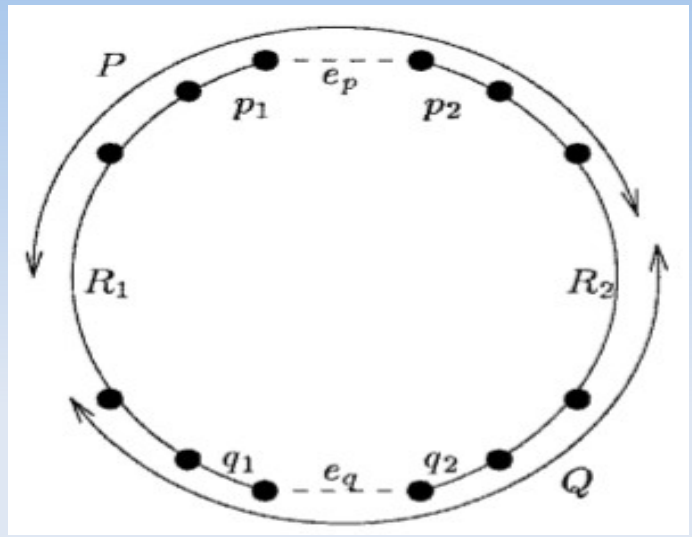
Lower Bound $\Omega(n \log n)$

I due insiemi contengono al più n/2 – 1 processori, sono contigui e disgiunti. Cosi il numero di messaggi ricevuti dai processori all'interno di uno dei due set è 1/2(n/2 – 1).

Sia σ_4 la più piccola sotto-sequenza di σ_4' , allora finché non c'è comunicazione tra i processori in P e in Q, $\sigma_1\sigma_2\sigma_3\sigma_4$ è uno scheduler. Dunque almeno almeno 1/2(n/2 – 1) messaggi sono ricevuti in σ_4 e per costruzione nessun messaggio è inviato tramite e_q.

Così $\sigma_1\sigma_2\sigma_3\sigma_4$ è un open scheduler.

Lower Bound $\Omega(n \log n)$



Ricapitolando: Siamo partiti con due scheduler per R_1 ed R_2 in cui erano inviati 2M(n/2) messaggi; successivamente abbiamo forzato l'anello in una configurazione inattiva e in fine forzato 1/2(n/2) messaggi addizionali facendo transitare i messaggi solo su uno tra e_q ed e_p ottenendo cosi un open scheduler in cui 2M(n/2)+ 1/2(n/2) messaggi vengono ricevuti.

Reti Sincrone

- Presenteremo ora un Upper e un Lower Bound per quanto riguarda le reti sincrone
- Gli algoritmi useranno gli identificativi in un modo inusuale
- Per l'upper bound presenteremo due algoritmi:
  - Il primo non unifome in cui tutti i processori partono nello stesso round
  - Il secondo uniforme in cui i processori possono partire in round differenti

Nonuniform Algorithm

Algoritmo:

Tutti i processori partono nello stesso round;

Si elegge a leader il processore con l'identificativo minore;

Lavorando in fasi divise in n round, nella fase i saranno inclusi i round da n*i+1 ad n*i+n;

Nella generica fase *i,* per ogni processore:

- Se il suo identificativo è *i*, allora tale processore invia un messaggio lungo l'anello e termina come leader.
- Se il suo identificativo non è *i* e riceve un messaggio, allora questo reinvia il messaggio al suo vicino e termina come non leader.

Nonuniform Algorithm

Esattamente il solo processore con il minimo identificativo viene eletto come leader, ed esattamente n messaggi sono inviati nella fase in cui tale leader viene trovato.

Il numero dei round dipende dunque dal minimo identificativo: sia esso m allora, l'algoritmo impiega n*(m+1) round per terminare. Ciò implica la **non uniformità** dell'algoritmo che ha la necessità di conoscere la grandezza dell'anello.

Complessità: O(n) messaggi inviati.

Uniform Algorithm

Non richiede la conoscenza della grandezza dell'anello.

I processori non partono necessariamente simultaneamente

Due idee fondamentali:

1) Un messaggio originato da un processore con identificativo *i* è ritardato 2^i -1 round da ogni processore che lo riceve prima di essere reinviato al prossimo processore in senso orario

2) Viene effettuata una fase iniziale di ”wake-up” in cui ogni processore che si sveglia spontaneamente invia un messaggio di ”wake up”. Un processore che riceve tale messaggio prima di essersi ”svegliato”, non partecipa alla computazione ma agirà esclusivamente come ripetitore.
In questa fase vengono ”eliminati” tutti i processori che non si sono svegliati spontaneamente.

Uniform Algorithm

Specifiche del funzionamento

- I messaggi inviati viaggiano a diverse velocità in base alla fase in cui si trovano.
- Un messaggio inizia nella prima fase e viaggia a velocità normale.
- Quando un messaggio arriva ad un processore partecipante entra nella seconda fase in cui viaggia con ritardo 2^i; per attuare tale rallentamento ogni processore ritarda l'invio del messaggio di 2^i-1 round.

Uniform Algorithm

Regole di blocco di un messaggio:

- Un **processore partecipante** blocca un messaggio se l'identificativo contenuto nel messaggio è maggiore del minimo identificativo incontrato sino a quel punto, **compreso** il proprio.
- Un **processore non partecipante**, blocca un messaggio se l'identificativo contenuto nel messaggio è maggiore del minimo identificativo incontrato sino a quel punto, **escluso** il proprio.

Dopo *n* round saranno spediti solo messaggi in seconda fase e le regole di blocco assicurano che un solo processore sia eletto a leader

Uniform Algorithm

Correttezza dell'algoritmo

Lemma: *Solo il processore con l'identificativo minore tra i partecipanti riceve indietro il suo messaggio.*

Dimostrazione: Sia p_i il processore con identificativo minore ed $<id_i>$ il messaggio da esso inviato: nessun processore può bloccare $<id_i>$.

Supponiamo per assurdo che un altro processore *pj* riceva il proprio messaggio $<id_j>$, allora $<id_j>$ è passato attraverso p_i e siccome $id_i<id_j$ e p_i è partecipante, allora p_i blocca $<id_j>$ che non può dunque arrivare a p_j

Uniform Algorithm

Complessità in base al numero dei messaggi inviati.

- Tre categorie di messaggi:
  - Prima fase
  - Seconda fase prima che il messaggio dell'eventuale leader entri nella sua seconda fase
  - Seconda fase dopo che il messaggio dell'eventuale leader entri nella sua seconda fase

Uniform Algorithm

<u>Lemma 1:</u> il numero totale dei messaggi della prima categoria è al più n.

Dimostrazione: al più un messaggio di prima fase è girato da ogni processore. Assumiamo per assurdo che un processore p_i giri due messaggi (nella loro prima fase) $<id_j>$ ed $<id_k>$. Assumiamo che p_j sia più vicino a p_i rispetto a p_k, così che $<id_k>$ deve passare attraverso p_j per arrivare a p_i. Cosi:

- O p_j si è gia svegliato e dunque ritarda $<id_k>$ facendolo entrare nella sua seconda fase, e dunque $<id_k>$ arriva a p_i come messaggio in seconda fase
- O p_j non invia il suo messaggio $<id_j>$

Contraddizione! $<id_k>$ ed $<id_j>$ non possono arrivare entrambi a p_i come messaggi della prima fase

Uniform Algorithm

Lemma 2: se p_j è a distanza (in senso orario) k da p_i allora un messaggio della prima fase è ricevuto da p_j non più tardi del round r+k.

Dimostrazione:

Caso base **k=1** è ovvio in quanto i vicini di p_i ricevono il messaggio $<id_i>$ nel round r+1.

Passo induttivo supponiamo che il processore a distanza k-1 da p_i riceve un messaggio di prima fase non più tardi del round r+k-1 allora:

- Se tale processore era già sveglio allora ha già inviato un messaggio di prima fase a p_j
- Altrimenti reinvia il messaggio $<id_i>$ di prima fase al processore p_j nel round r+k.

Uniform Algorithm

Lemma 3: il numero dei messaggi nella seconda categoria è al più n.

Dimostrazione: dopo r+n round tutti i messaggi passano nella loro seconda fase. Il messaggio del leader infatti entra nella sua seconda fase al più n round dopo l'invio del primo messaggio. I messaggi della seconda categoria vengono inviati negli n round successivi al round in cui il primo processore si è svegliato. Di conseguenza siccome un messaggio <i> è ritardato per 2^i-1 round sarà inviato $n/2^i$ volte in questa categoria. Il numero massimo si ottiene nel caso in cui tutti i processori partecipano, e gli identificativi sono i più piccoli possibili cosi che il numero totale di messaggi nella seconda categoria sia $\sum_{i=1}^{n-1} \frac{n}{2^i} \leq n$

Uniform Algorithm

Lemma 4: nessun messaggio viene girato una volta che <id_i> torni a p_i

Lemma 5: il numero totale di messaggi della terza categoria è al più 2n.

Dimostrazione:

- sia p_i l'eventuale leader e p_j un altro processore partecipante (cosi id_i<id_j)
- non ci sono messaggi una volta che <id_i>torna a p_i
- <id_i> è rallentato al più 2^{id} volte da ogni processore quindi ritorna al processore p_i al più dopo $n*2^{idi}$ round
- I messaggi della terza categoria sono inviati solo durante gli $n*2^{idi}$ round

Uniform Algorithm

- Durante tali round il messaggio generico <id_j> vine girato al più $\frac{1}{2^{id_j}} \cdot n \cdot 2^{id_i} = n \cdot 2^{id_i - id_j}$ volte.

  E dunque il numero massimo di messaggi trasmessi in questa categoria sono al più $\sum_{j=0}^{n-1} \frac{n}{2^{id_j - id_i}}$

  Tale quantità è minore o al più uguale a 2n

  Unendo i lemma 1,3 e 5 si ottiene:

  Teorema: *Esiste un algoritmo sincrono per la leader election in cui la complessità di messaggi è al più 4n. Dunque una complessità in termini di messaggi pari a O(n).*

Lower Bound

I due algoritmi presentati per poter ottenere complessità di $O(n)$ messaggi, hanno richiesto due **proprietà indesiderabili**:

- utilizzano gli identificativi in modo inusuale
- fanno dipendere il numero dei round dagli identificativi assegnati ai processori.

Tali proprietà sono inerenti ad ogni algoritmo efficiente: infatti se un algoritmo è forzato ad usare gli identificativi solo per la comparazione allora richiederà l'invio di $\Omega(n \log n)$ messaggi

Comparison-Based Alg

- Tutti i processori partono nello stesso round.
- Un anello è specificato dall'elencazione degli identificativi in senso orario iniziando dal più piccolo.
- Una **esecuzione ammissibile** è completamente **definita dalla configurazione iniziale** perchè non c'è scelta sul rallentamento dei messaggi nè sull'ordine dei passi dei processori
- A sua volta la configurazione iniziale è definita dall'anello stesso

Comparison-Based Alg

Notazione:

- *exec(R):* l'ammissibile esecuzione determinata dall'anello R,quando le scelte effettuate da un algoritmo sono chiaramente deducibili dal contesto

- *Matching:* due processore pi e pj in due anelli differenti ”matchano” se hanno la stessa posizione nei rispettivi anelli (stessa distanza dal processore con identificativo minore).

- *Order-equivalent:* due anelli $x_0 \ldots x_{n-1}$ ed $y_0 \ldots y_{n-1}$ sono order-equivalent se per ogni i e j si ha che $x_i < x_j$ sse $y_i < y_j$

Comparison-Based Alg

Def: *Un algoritmo è **comparison based** se ”si comporta nello stesso” modo su anelli che hanno lo stesso modello di ordine per gli identificativi.*

Definiamo cosa significa ”si comporta nello stesso modo”:

Saremmo tentati di dire che un algoritmo è comparison based se su anelli o*rder-equivalent* invia gli stessi messaggi è prende le stesse decisioni; ma siccome diversi anelli possono avere diversi identificativi, tale affermazione risulterebbe errata. Pertanto concentriamo la nostra attenzione sul modello dei messaggi, cioè quando e come i messaggi sono inviati, più tosto che sul loro contenuto.

Comparison-Based Alg

- Consideriamo dunque due possibili esecuzioni δ_1 ed δ_2 e due processori p_i e p_j, diremo che il comportamento di p_i in δ_1 è **simile** nel round k al comportamento di p_j in δ_2 se:
  - p_i invia un messaggio al suo vicino destro(o sinistro) nel round k in δ_1 se e solo se p_j invia un messaggio al suo vicino destro(o sinistro) nel round k in δ_2
  - p_i termina come leader nel round k in δ_1 se e solo se p_j termina come leader nel round k in δ_2

il comportamento di p_i in δ_1 è simile al comportamento di p_j in δ_2, se le condizioni appena mostrate valgono per ogni $k \geq 0$.

Comparison-Based Alg

Definizione:

Un algoritmo è comparison based se per ogni coppia di order-equivalent anelli R_1 de R_2, ogni coppia di processori ”matchanti” *ha lo stesso comportamento in exec(R_1) ed exec(R_2)*

Lower Bound per C.B.A

Si prende in considerazione un anello fortemente simmetrico in cui sono presenti molti order-equivalent vicinati. Si procederà eseguendo A su tale anello e richiedendo che quando un processore invia un messaggio allora tutti i processori con vicinato order-equivalent a tale processore, inviino un messaggio nello stesso round

Definizione: *Un round r è* ***attivo*** *in un esecuzione per un anello R se qualche processore invia un messaggio nel round r dell'esecuzione. Se R è derivabile dal contesto denotiamo con* r_k *l'indice del k-esimo round attivo.*

Siccome l'algoritmo è comparison-based, dati due anelli R_1 ed R_2 order-equivalent allora un round è attivo in exec(R_1) sse è attivo anche in exex(R_2). Lo stato di un processore dopo k round attivi dipende esclusivamente dal suo k-vicinato.

Lower Bound per C.B.A

Lemma 1: Siano R_1 ed R_2 due anelli order-equivalent, e siano p_i in R_1 e p_j in R_2 due processori con identico k-vicinato. Allora la sequenza di transizioni effettuate da p_i nei round da 1 a r_k in exec(R_1), è la stessa effettuata da p_j negli stessi round in exec(R_2).

- Verrà provato per induzione che dopo k round attivi un processore può ottenere informazione solo sui processori che sono distanti al più k da esso.
- Caso base **k=0** due processori con lo stesso 0-vicinato hanno lo stesso identificativo quindi sono identici

Lower Bound per C.B.A

Passo induttivo: Assumendo che due processori p_i e p_j abbiano lo stesso (k-1)-vicinato allora si troveranno nello stesso stato dopo (k– 1) round attivi. Inoltre i rispettivi vicini hanno lo stesso (k–1)vicinato; di conseguenza anche i loro vicini saranno nello stesso stato dopo il (k–1)-esimo round attivo. Al k-esimo round attivo se p_i riceve un messaggio da un suo vicino allora anche p_j deve ricevere un messaggio dal suo vicino in quanto i vicini di p_i e p_j sono nello stesso stato di conseguenza al termine del k-esimo round attivo p_i e p_j saranno nello stesso stato. Per completezza si noti che nei round non attivi tra il (k-1)-esimo round attivo e il k-esimo round attivo siccome p_i e p_j hanno la stessa funzione di transizione se uno dei due cambia stato allora entrambi cambieranno stato.

Lower Bound per C.B.A

Da k-vicinato a **order-equivalent k-vicinato**.

(l'algoritmo è comparison-based e richiede che l'anello R su cui l'algoritmo è eseguito sia ”spaziato”)

Definizione di ”spaziato”:

Per ogni identificativo x nell'anello gli identificativi da *x-1* ad *x-n* non sono usati nell'anello

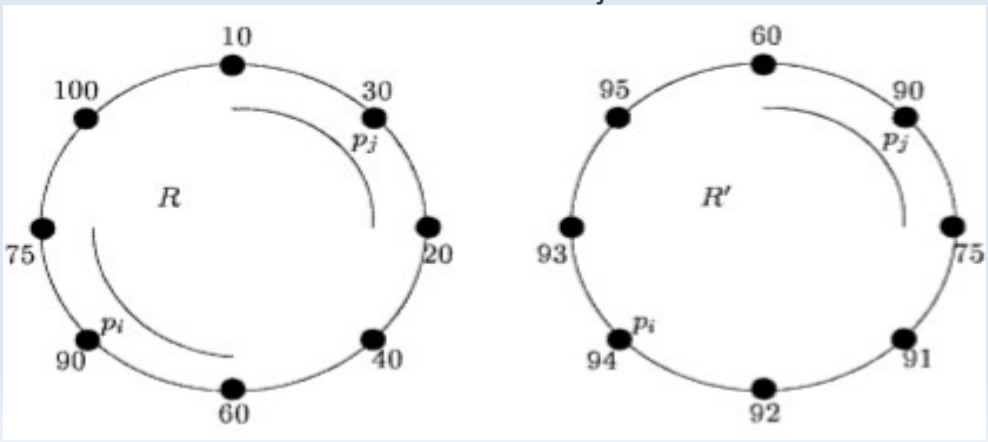
(per ogni due identificativi di un anello ci sono n identificativi inutilizzati tra loro)

Lower Bound per C.B.A

Lemma 17: Sia R un anello spaziato e siano pi e pj due processori con order-equivalent k-vicinato allora p_i e p_j hanno lo stesso comportamento nei round da 1 ad r_k in exec(R).

Dimostrazione: Costruiamo un altro anello R' tale che:

- p_j in R' ha lo stesso vicinato di p_i in R
- gli identificativi sono univoci
- R' è order-equivalent ad R e p_j in R' ”matcha” con p_j in R.



Lower Bound per C.B.A

Nei round da 1 a r_k le transazioni effettuate da p_i in exec(R) sono le stesse effettuate da p_j in exec(R') cosi p_i e p_j hanno lo stesso comportamento nelle rispettive esecuzioni.
Siccome p_j in R' ”matcha” con p_j in R, allora il comportamento di p_j nei round da 1 ad r_k è simile al comportamento di p_i nei round da 1 a r_k C.V.D.

Teorema: *Per ogni n≥8 che è potenza di 2 esiste un anello S_n, di grandezza n, tale che per ogni algoritmo sincrono comparison-based A Ω(n log n) messaggi sono inviati in un ammissibile esecuzione di A su S_n*

Dimostrazione:

- Si basa sulla costruzione di S_n che sarà altamente simmetrico

Lower Bound per C.B.A

- Si definisce un anello R di n processori in cui per ogni processore p_i l'identificativo del processore sarà *rev(i),* dove *rev(i)* è l'intero la cui rappresentazione binaria usa log*n* bit ed è il riverso della rappresentazione binaria di i. Esempio p_1 $(1)_2$=001 di conseguenza rev(i)=$(100)_{10}$=4
- Si suddivide R in segmenti consecutivi di lunghezza j (potenza di 2)
- S_n sarà la versione ”spaziata” di R ottenuta moltiplicando gli identificativi di R per n+1 e aggiungendo n.
- Resta ora da capire quanti vicinati order-equivalent esistono in S_n per una data size, questo è mostrato nel seguente lemma il cui risultato sarà utilizzato successivamente per mostrare un lower bound per il numero dei round e per il numero dei messaggi

Lower Bound per C.B.A

Lemma 19: Per ogni $k < n/8$ e per tutti i k-vicinati N di S_n ci sono più di $n/2(2k+1)$ k-vicinati di S_n che sono order-equivalent ad N (N incluso)

Dimostrazione: N è una sequenza di 2k+1 identificativi. Sia j la più piccola potenza di 2 maggiore di 2k+1: si partizioni S_n in n/j segmenti consecutivi tali che uno di questi segmenti copra tutto N. Tutti questi segmenti sono order-equivalent in modo tale che ci siano almeno n/j vicinati order-equivalent ad N. Siccome $j<2(2k+1)$, allora il numero di vicinati order equivalent ad N sono più di n/2(2K+1).

Come detto in precedenza dal lemma deriviamo i seguenti 2:

Lower Bound per C.B.A

Lemma 20: il numero di round attivi in exec(S_n) è almeno n/8

Dimostrazione: Sia T il numero di round attivi. Supponiamo per assurdo che T<n/8. Sia p_i il leader in *exec(S_n)* dal lemma19 ci sono più di n/2(2T+1) T-vicinati order-equivalent al T-vicinato di p_i. Dall'assunzione otteniamo che:

$$\frac{n}{2(2T+1)} > \frac{n}{2(2n/8+1)} = \frac{2n}{n+4}$$

Per **n≥8**, 2n/(n+4) è maggiore di 1. Esiste dunque un altro processore p_j il cui T-vicinato è order-equivalent al T-vicinato di p_i, che per il lemma17 diventa anch'esso leader contraddicendo la correttezza di A.

Lower Bound per C.B.A

Lemma21: Almeno n/2(2k+1) messaggi sono inviati nel k-esimo round attivo in exec(S_n) per ogni $1 \leq k \leq n/8$

Dimostrazione: Finchè si è in un round attivo almeno un processore invia un messaggio. Per il lemma19 ci sono più di n/2(2k+1) processori il cui k-vicinato è order-equivalent al k-vicinato di p_i. Per il lemma17 ogni uno di essi invia un messaggio nel k-esimo round attivo.

Dal lemma20 e 21 deriviamo dunque che il numero di messaggi in exec(S_n) è:

$$\sum_{k=1}^{n/8} \frac{n}{2(2k+1)} \geq \frac{n}{6} \sum_{k=1}^{n/8} \frac{1}{k} > \frac{n}{6} \ln \frac{n}{8}$$

che è un $\Omega(n \log n)$

C.V.D

Coclusione

Domande?