Coordinazione distribuita

Vines Giuseppina
Colombo Verusca

Introduzione

- Meccanismi che permettono la sincronizzazione delle azioni dei processi (ordinamento degli eventi)
- Atomicità: proprietà dei vari modi di esecuzione
- Metodi per la risoluzione delle situazioni di stallo

Ordinamento degli eventi

- Problema:
  - tutti i sistemi della rete devono affermare che un evento a di un sistema i si è svolto prima o dopo dell'evento b del sistema j.
- Ostacoli:
  - ritardo tra l'effettivo verificarsi dell'evento e il momento in cui gli altri sistemi lo osservano;
  - Variazione della lettura dei clock dei vari sistemi per la mancanza di sincronizzazione

Ordinamento degli eventi

- Soluzione:

In un sistema distribuito, i processi interagiscono tramite lo scambio di messaggi.

Supponiamo che un evento sia il momento in cui un processo invia o riceve un messaggio e che un messaggio si può ricevere solo dopo essere stato inviato, la relazione VERIFICATO PRIMA si può definire:

Relazione VERIFICATO PRIMA

- Se a e b ϵ Pi, allora a è stato eseguito prima di b quindi a→b
- Se a corrisponde all'evento dell'invio di un messaggio di P1 e b alla ricezione di quel messaggio da parte di P2 allora a→b
- Se a→b e b→c allora a→c

N.B.:non può accadere che a→a(non riflessivo)

Relazione VERIFICATO PRIMA

- Se a non→b e b non→a allora a e b sono concorrenti, quindi non influiscono in modo causale l'uno sull'altro,ciò significa che non è possibile sapere quale tra di loro è verificato prima.

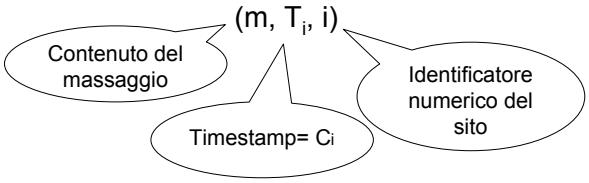
Ordinamento totale

La relazione VERIFICATO PRIMA ordina solo parzialmente gli eventi, in quanto,come abbiamo già detto, in sistemi distribuiti manca un clock comune o una sincronizzazione perfetta.

Per risolvere questo problema,ad ogni evento si associa una marca temporale (timestamp), all'interno, poi, di ciascun processo si definisce un clock logico, che corrisponde alla marca temporale del proprio evento

Ordinamento totale

Quando un sistema trasmette un messaggio, incrementa il clock di uno ed invia il messaggio nella forma:



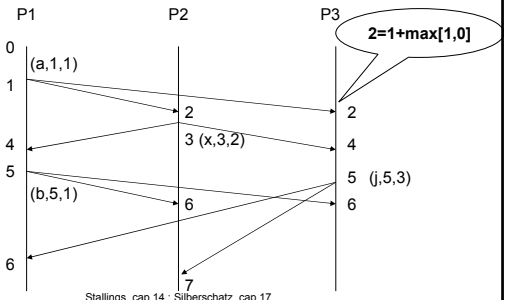
Ordinamento totale

- Quando un messaggio viene ricevuto, il sistema ricevente j aggiorna il proprio clock a :

  $C_j = 1 + \max[C_j, T_i]$
- Il tempo associato ad ogni messaggio è il timestamp dello stesso e l'ordinamento di questi tempi è dato dalle seguenti regole:
  - $T_i < T_j$
  - $T_i = T_j$ e i<j
- Se due messaggi hanno lo stesso timestamp,vengono ordinati in base al numero dei loro siti

Ordinamento totale

Esempio del funzionamento del timestamp:



Mutua esclusione distribuita

- Quando 2 o più processi concorrono per l'uso di risorse di sistema,occorre garantire che una alla volta si trovi nella sezione critica(porzione di programma che utilizza una risorsa non condivisibile detta anch'essa "critica")
- Gli algoritmi per la mutua esclusione possono essere di tipo centralizzato e di tipo distribuito.

Metodo centralizzato

- Dati n processi, se ne sceglie uno per coordinare l'accesso alle sezioni critiche.
- I processi che necessitano della mutua esclusione, inviano un messaggio di richiesta al coordinatore, che o invia un messaggio di risposta o accoda la richiesta.
- Quando il coordinatore riceve un messaggio di rilascio, preleva una richiesta dalla coda e invia il messaggio di risposta al processo richiedente.

Metodo centralizzato

- Vantaggi
  - Diretto
  - Semplice
  - Assicura la mutua esclusione
  - Non c'è alcuna possibilità d'attesa,il controllore non soddisfa alcuna richiesta finché la risorsa critica non viene rilasciata

Metodo centralizzato

- Svantaggi
  - Se il controllore si guasta, il meccanismo di mutua esclusione si interrompe fino a quando un nuovo processo non lo sostituisce.
  - Una volta eletto, il nuovo processo coordinatore deve interrogare ciclicamente tutti i processi per ripristinare la coda delle richieste.

Metodo distribuito

- Quando un processo vuole entrare nella propria sezione critica, genera una nuova marca temporale e invia il messaggio *request (Pi,Ti)* a tutti i processi(esso compreso).
- Un processo che riceve tale messaggio ha due possibilità: o rispondere immediatamente o differire l'invio della risposta. Questa decisione dipende da tre fattori…

Metodo distribuito

- Pi proroga la risposta a Pj, se si trova nella propria sezione critica
- Pi risponde a Pj, se ***non*** vuole entrare nella propria sezione critica
- Se Pi vuole entrare nella propria sezione critica, ma non vi è ancora entrato, confronta la sua marca temporale Ti con quella della richiesta di Pj. Se Ti>Tj allora risponde immediatamente a Pj,altrimenti differisce la risposta

Metodo distribuito

- Una volta ricevuto il messaggio di risposta da tutti gli altri processi del sistema, Pi entra nella propria sezione critica, differendo le richieste in arrivo, fino all'uscita dalla sezione critica.

Metodo distribuito

- Vantaggi
  - Mutua esclusione
  - Assenza di situazioni di stallo
  - Assenza di attese indefinite
  - Il numero di messaggi di processi indipendenti e concorrenti risultano pari a 2(n-1) cioè il minimo richiesto

Metodo distribuito

- Svantaggi
  - I processi devono conoscere tutti i processi del sistema. Quando un nuovo processo si aggiunge al gruppo che partecipano alla mutua esclusione, deve:
    1. Ricevere i nomi di tutti i componenti del gruppo;
    2. Essere distribuito a tutti gli altri processi del gruppo.
  - Se uno dei processi si interrompe per un errore,crolla tutto lo schema.
  - I processi che si trovano nelle proprie sezioni critiche,devono interrompersi spesso per informare quelli che intendono entrare nelle proprie sezioni critiche.

Metodo token-passing

Un approccio diverso alla mutua esclusione è quello di far girare tra i processi del sistema un contrassegno (*token*), uno speciale tipo di messaggio che permette di entrare nella sezione critica.

In un sistema esiste solo un contrassegno che può essere posseduto solo da un processo alla volta.

Per realizzare la mutua esclusione, si fa passare il token attraverso una struttura logica ad anello.

Quando il processo esce dalla propria sezione critica, il contrassegno viene rimesso in circolazione e viene accettato dal processo successivo che vuole entrare nella sezione critica.

Metodo token-passing

L'algoritmo può essere così rappresentato:

- *token*= array il cui k-esimo elemento è il timestamp dell'ultima volta che il token ha visitato Pk.
- *richesta*= array il cui j-esimo elemento è il timestamp dell'ultima volta che il token ha visitato Pj

Metodo token-passing

```
if not token_presente then begin clock:= clock +1;
broadcast(Richiesta, clock, i);
      wait(accesso, token);
      token_presente:=true;
      end
endif;
      token_trattenuto:=true;
      <Sezione critica>
      token(i):=clock;
      token_trattenuto:=false;
      for j :=i+1 to n, 1 to i—1 do
if (richiesta(j)>token(j)) ^ token_presente
       then begin
        token_presente:= false;
        send(accesso, token(i))
       end
endif;
      when received(Richiesta, k, j) do
       richiesta(j):=max(richiesta(j),k);
        if token_presente ^ not token_trattenuto then
        <testo della conclusione>
       endif
      enddo
```

Introduzione

A

Conclusione

B

Parte A

- Gestisce l'uso della sezione critica e si compone di due parti:
  - Introduzione:il token viene assegnato arbitrariamente ad un processo che se vuole può entrare nella sezione critica.
  - Conclusione:se il processo vuole entrare nella sezione critica e non ha il token, invia la Richiesta con timestamp e aspetta fino a quando Pj,uscito dalla sezione critica trasmette il token al primo processo che nell'array *richiesta* tale che *richiesta(k)>token(k)*

Parte B

- Riguarda le azione intraprese al ricevimento di una Richiesta;
- La variabile clock rappresenta un contatore locare per il calcolo del timestamp;
- L'operazione wait(accesso,token) mette il processo in attesa finché non riceve il messaggio di tipo accesso che viene inserito nell'array token.

Metodo token-passing

- Vantaggi
  - Assenza di attesa indefinita(se anello unidirezionale)
  - Il numero di messaggi varia da n(alta contesa)a infinito(bassa contesa)
- Svantaggi
  - Due tipi di guasti:
    1. Se il contrassegno va perduto, occorre indire un'elezione per la generazione di uno nuovo
    2. Se un processo si interrompe a causa di un errore,occorre ripristinare un nuovo anello logico

Atomicità

- Transazione atomica: unità di programma che si deve eseguire in modo atomico,cioè tutte le operazioni ad essa associate devono essere portate tutte a compimento altrimenti nessuna di esse viene eseguita.
- Più siti possono partecipare ad una singola transazione,quindi se si verifica un guasto in uno di questi siti o nella linea di comunicazione che li connette può causare una valutazione errata.
- Il coordinatore delle transazioni di un sistema distribuito deve,quindi, assicurare che ogni transazione eseguita deve rispettare l'atomicità.

Atomicità

- Per ogni transazione il coordinatore è responsabile di:
  - Avviare l'esecuzione della transazione;
  - Scomporre la transazione in sottotransazioni e distribuirle ai siti appropriati per l'esecuzione;
  - Coordinare la conclusione della transazione(successo o fallimento)

Protocollo di conferma a 2 fasi

- Per soddisfare la terza proprietà,il coordinatore deve eseguire un protocollo di conferma.
- Tra quelli più semplici ed utilizzati analizzeremo il protocollo di conferma a due fasi (two phase commit protocol-2PC)
- Sia *T* una transazione avviata nel sito *Si,* e sia Ci il coordinatore in *Si.* Quando *T* completa la propria esecuzione Ci attiva il protocollo 2PC.

Protocollo di conferma a 2 fasi

- Fase **1. :**
  - Ci aggiunge l'elemento **<*T* prepare>** al giornale delle modifiche, lo registra nella memoria stabile e lo spedisce a tutti i siti nei quali T è in esecuzione.
  - Ricevuto il messaggio, il gestore delle transazioni in ciascuno di tali siti determina se il sito è pronto a terminare con successo la sua parte di *T*.
  - Se la risposta è *no,* il gestore delle transazioni aggiunge un elemento **<Tno >** al giornale delle modifiche e risponde con il messaggio **abort(*T)*** a *Ci.*
  - Se la risposta è *si,* aggiunge l'elemento **<T ready>** al giornale delle modifiche,registra nella memoria stabile tutti gli elementi del giornale delle modifiche corrispondenti a T e risponde a *Ci,* con il messaggio **ready(*T)***.

Protocollo di conferma a 2 fasi

- Fase 2:
  - Una volta ricevuta la risposta al messaggio **prepare(*T)*** a esso inviata da tutti i siti, o dopo che è trascorso un predeterminato intervallo di tempo dall'invio del messaggio, Ci può determinare se la transazione T può essere chiusa con successo o è fallita.
  - Secondo il verdetto, uno tra gli elementi <T commit> o <T abort> viene aggiunto al giornale delle modifiche e registrato nella memoria stabile.
  - Di conseguenza il coordinatore invia ai siti partecipanti uno tra i messaggi commit(*T)* o *abort(*T), i quali lo registrano nel proprio giornale delle modifiche.

Protocollo di conferma a 2 fasi

- Un sito nel quale *T è* in esecuzione può interrompere incondizionatamente l'esecuzione della transazione solo prima di avere spedito il messaggio ready(*T)* al coordinatore. In effetti questo messaggio è una promessa di eseguire l'ordine del coordinatore di terminare con successo T o farlo fallire.
- Il successo della transazione richiede l'unanimità, perciò il destino di T è deciso non appena almeno un sito risponde con un abort(*T).* Essendo anch'esso coinvolto nell'esecuzione, il sito di coordinamento Si, può decidere in modo unilaterale d'interrompere l'esecuzione di *T.*
- Il verdetto finale su *T* è determinato quando il coordinatore scrive la decisione (commit o abort) nel giornale delle modifiche e nella memoria stabile.

Gestione dei guasti nel 2PC

- Uno dei principali svantaggi del protocollo consiste nel fatto che il verificarsi di un guasto nel coordinatore può causare il blocco dell'intero sistema, poiché potrebbe essere necessario rinviare la decisione di chiusura, fino al momento in cui *Ci,* riprenderà il normale funzionamento.
- Distinguiamo 3 tipi di guasto:
  - Guasto di un sito partecipante
  - Guasto del coordinatore
  - Guasto della rete

Guasto di un sito partecipante

- Quando un sito partecipante Sk si riprende da un guasto deve innanzi tutto esaminare il proprio giornale delle modifiche per determinare la sorte delle transazioni che erano in esecuzione quando si è guastato.
- Sia T una di queste transazioni, si devono considerare i casi possibili:
  - Il giornale non ha alcun elemento, il sito esegue undo(*T).*
  - Il giornale delle modifiche ha un elemento <*T* commit>, il sito esegue redo(*T).*
  - Il giornale delle modifiche ha un elemento <*T* abort>, il sito esegue undo(*T).*
  - Il giornale delle modifiche ha <*Tready>,* il sito deve consultare *Ci* per decidere la sorte di *T.*

Guasto di un sito partecipante

- Se *Ci* è attivo, notifica a Sk il successo eseguendo redo(*T),* o il fallimento undo(*T).*
- Se non è attivo, Sk deve cercare di risalire alla sorte di T dagli altri siti, inviando a ciascuno di essi il messaggio query—status(*T).* Quando un sito riceve questo messaggio, esamina il proprio giornale delle modifiche per capire se ha eseguito T e com'è terminato; quindi notifica il risultato della ricerca a Sk.
- Se nessuno contiene informazioni sullo stato di T,Sk non è in grado di chiuderlo e rimanda la decisione fino a quando non sarà ripristinato il sito che contiene le notizie richieste

Guasto del coordinatore

- Se durante l'esecuzione del protocollo di conferma per la transazione *T,* si verifica il malfunzionamento del coordinatore, la sorte di *T* deve essere decisa dai siti partecipanti all'esecuzione. Anche in questo caso si distinguono varie situazioni…

Guasto del coordinatore

- Se un sito attivo contiene l'elemento <*Tcommit*> nel proprio giornale delle modifiche , T è terminata con successo.
- Se un sito attivo contiene l'elemento <*T* abort> nel proprio giornale delle modifiche, T è fallita.
- Se alcuni tra i siti attivi non contengono l'elemento <T ready> nel proprio giornale, il coordinatore guasto *Ci* non può aver deciso il successo di *T* perché, se un sito non contiene nel proprio giornale l'elemento <*T* ready> non può avere spedito il messaggio ready (*T) a Ci.* D'altra parte, il coordinatore potrebbe averne deciso il fallimento. Piuttosto che attendere il ripristino di Ci è preferibile l'interruzione dell'esecuzione di T.

Guasto del coordinatore

- Se non si presenta nessuno dei casi precedenti, tutti i siti attivi devono contenere nel proprio giornale l'elemento <*T* ready>, senza però altri elementi di controllo (come <*T abort*> o <*T* commit>). Poiché il coordinatore si è guastato, *è* impossibile stabilire prima del suo ripristino se questi avesse preso una decisione o quale essa fosse; quindi i siti attivi devono attendere il ripristino di Ci.
- Poiché la sorte di T *è* dubbia, può succedere che T continui a detenere risorse del sistema.

Guasto del coordinatore

- Ad esempio, *T* può continuare a detenere un diritto d'accesso bloccante sui dati contenuti nei siti attivi. Questa *è* una situazione indesiderabile poiché possono trascorrere ore o giorni prima che torni in attività. Durante questo periodo altre transazioni potrebbero essere costrette ad attendere il completamento di T.
- Ne segue che i dati non saranno disponibili non solo nel sito guasto *(Ci)* ma anche nei siti attivi. Il numero di dati non disponibili aumenta col crescere del periodo di inattività di *Ci*. Questa situazione prende il nome di problema del *bloccaggio,* poiché *T*è bloccata nell'attesa del ripristino del sito .

Guasto della rete

- Quando il guasto riguarda un collegamento di una rete, tutti i messaggi instradati in esso non arrivano intatti alla destinazione. Anche qui sono applicabili gli schemi appena proposti.
- Se riguarda più collegamenti, si può verificare un partizionamento della rete stessa. In questo caso ci sono due possibilità:
  - il coordinatore e tutti i suoi partecipanti restano nella stessa partizione, in questo caso il guasto non ha alcun effetto sul protocollo di conferma
  - coordinatore e partecipanti si ritrovino in partizioni diverse; in questo caso si riduce al caso del guasto di un collegamento.

Controllo della concorrenza

Il gestore delle transazioni di una base di dati distribuita gestisce l'esecuzione delle sottotransazioni che accedono ai dati di un sito locale. Ciascuna di queste transazioni può essere:

- una transazione locale se è in esecuzione solamente in quel sito
- parte di una transazione globale se coinvolge più siti

Ciascun gestore delle transazioni è responsabile del mantenimento di un giornale delle modifiche per le attività di ripristino e della partecipazione.

Protocolli d'accesso bloccante

- Concetti:
  - **Modo d'accesso bloccante condiviso:** se Ti accede all'elemento Q bloccandolo in modo che può leggere, ma non scrivere in esso;
  - **Modo d'accesso bloccante esclusivo:** se Ti accede a Q bloccandolo in modo che può leggere e scrivere in esso;

Protocolli d'accesso bloccante

Schema generale:

- Ti,per accedere a Q, deve prima bloccarlo nel modo più appropriato:
  - Se Q è libero, la realizzazione del blocco da parte di Ti è garantita;
  - Se Q è bloccato, allora:
    - Se Ti richiede un blocco esclusivo, deve attendere il rilascio del blocco di Q;
    - Se Ti richiede un blocco condiviso, deve attendere il rilascio del blocco di Q solo se questo si trova in blocco esclusivo;

Protocolli d'accesso bloccante

Schema senza replicazione:

- Ogni sito ha un gestore dei diritti d'accesso bloccante che si occupa delle richieste d'accesso e di rilascio per i dati di quel sito.
- Se T vuole bloccare Q ϵ Si, invia un messaggio al gestore dei diritti d'accesso bloccante di Si dove richiede il tipo d'accesso che vuole.
- La richiesta verrà ritardata fino a quando potrà essere soddisfatta e il gestore risponderà al richiedente.

Protocolli d'accesso bloccante

Schema senza replicazione:

- Vantaggio:
  - Facilmente realizzabile (solo 2 messaggi)
- Svantaggio:
  - La gestione delle situazioni di stallo è più difficile, perché le richieste di bloccaggio e di rilascio non vengono inoltrate allo stesso sito.

Protocolli d'accesso bloccante

Protocollo di maggioranza:

- È una modifica dello schema precedente:
- Quando T vuole bloccare Q, replicato in n siti differenti,deve inviare una richiesta alla maggior parte di questi siti.
- Quando la maggioranza di gestori a cui è stata inviata la richiesta, determinerà che la richiesta è stata soddisfatta, la transazione richiedente può operare su Q.

Protocolli d'accesso bloccante

Protocollo di maggioranza:

- Vantaggi:
  - Gestione dei dati in modo decentralizzato
- Svantaggi:
  - Realizzazione (2(n/2 +1) + (n/2 + 1) messaggi)
  - Gestione delle situazioni di stallo (le richieste non vengono inoltrate ad uno stesso sito)

Protocolli d'accesso bloccante

Protocollo sbilanciato:

- Come il modello precedente,differisce per il fatto che gestisce in maniera diversa le richieste d'accesso condivise da quelle esclusive:
  - Accesso bloccante condiviso:T invia la richiesta al gestore del sito dove c'è una copia di Q .
  - Accesso bloccante esclusivo:T invia la richiesta a tutti i gestori dei siti che contengono una replica di Q.

Protocolli d'accesso bloccante

Protocollo sbilanciato:

- Vantaggi:
  - Minor carico sulle operazioni di lettura
- Svantaggi:
  - Maggior carico sulle operazioni di scrittura
  - Gestione dello stallo

Protocolli d'accesso bloccante

Metodo con coordinatore singolo:

- Questo schema impiega un singolo gestore di un solo sito *Si* in tutto il sistema.
- Le richieste di blocco vengono inviate tutte a Si che decide quali si possono essere soddisfatte e quindi inviare una risposta, e quali devono essere differite.
- La transazione può leggere Q da qualsiasi sito che contiene una sua copia,ma nel caso di scrittura, devono essere coinvolti tutti i siti che ne hanno una copia

Protocolli d'accesso bloccante

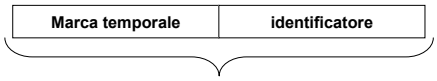
Metodo con coordinatore singolo:

- Vantaggi:
  - Semplice realizzazione(2 messaggi);
  - Semplice gestione di stallo (solo1gestore in un sito);
- Svantaggi:
  - Strozzatura (elaborazione tutte le richieste)
  - Vulnerabilità (Si guasto)
- Compromesso: Coordinatori multipli

Uso delle marche temporali

Ci sono due modi per generare marche temporali uniche:

1. Centralizzato: si sceglie un singolo sito che può usare o un contatore logico o il suo clock locale per distribuire marche temporali
2. Distribuito: la marca temporale unica globale è data dall'unione della marca temporale unica locale con l'identificatore del sito



Identificatore unico globale

Uso delle marche temporali

Come per l'ordinamento degli eventi, per garantire la sincronizzazione tra i vari clock logici, si richiede che il sito Si incrementi il proprio clock logico LCi ogni volta che Ti con marca temporale <x,y>(dove x>LCi) visita quel sito.

Stallo distribuito

- Stallo: blocco permanente di un insieme di processi che competono per ottenere risorse del sistema o comunicano
- Nei sistemi distribuiti presenta problematiche più complesse:
  - Nessun nodo ha una conoscenza accurata dello stato del sistema nel suo insieme
  - Ogni scambio di messaggi è affetto da un ritardo imprevedibile
- Due tipi di stallo distribuito:
  - Stallo provocato dall'allocazione di risorse:
    - Ogni processo di un'insieme di processi richiede una risorsa posseduta da un altro processo dell'insieme
  - Stallo legato alla comunicazione di messaggi:
    - I messaggi sono le risorse attese dai processi
    - Ogni processo dell'insieme è in attesa di un messaggio da un altro processo dello stesso insieme, e nessun processo invia messaggi

Stallo nell'allocazione delle risorse

- Esiste uno stallo delle risorse solo se valgono ***tutte*** le seguenti condizioni:
  1. Mutua esclusione: solo un processo alla volta può usare una risorsa
  2. Possesso e attesa: un processo può tenere allocate le risorse, mentre aspetta l'assegnazione delle altre
  3. Assenza di prerilascio: nessuna risorsa può essere ritirata da un processo che la possiede
  4. Attesa circolare: esiste una catena chiusa di processi tale che ogni processo possiede almeno una risorsa di cui il processo successivo nella catena ha bisogno
- Scopo di un algoritmo che gestisce lo stallo è evitare la formazione dell'attesa circolare, o rilevare il suo reale o potenziale verificarsi

Prevenzione dello stallo

- Gli algoritmi per prevenire ed evitare le situazioni di stallo in un sistema centralizzato, opportunamente modificati, si possono usare anche in un sistema distribuito
  - Es.: la condizione possesso e attesa può essere prevenuta obbligando un processo a richiedere tutte le risorse di cui ha bisogno in una volta, e bloccando il processo finché tutte le sue richieste non possono essere soddisfatte simultaneamente
    - Approccio inefficiente:
      - Il processo può attendere molto tempo prima che tutte le risorse siano liberate, mentre avrebbe potuto procedere con solo alcune di esse
      - Le risorse allocate ad un processo possono rimanere inutilizzate per lunghi periodi, durante i quali non possono venire assegnate a nessun altro processo

Tecnica di prevenzione basata sull'ordinamento delle risorse

- Si può usare definendo un ordinamento *totale* tra le risorse del sistema:
  - alle risorse si assegnano numeri unici
  - un processo può richiedere una risorsa con numero unico i, su qualsiasi unità di elaborazione, solo se non possiede una risorsa con un numero unico >i
- Facile da realizzare e elaborazione poco onerosa
- Previene la condizione di attesa circolare *
- Svantaggio: le risorse possono venire richieste in un ordine diverso da quello in cui vengono utilizzate, quindi possono venire trattenute più a lungo del necessario *

Algoritmo del banchiere

- Si sceglie uno dei processi del sistema (il *banchiere*) che mantiene le informazioni necessarie per eseguire l'algoritmo
- Ogni richiesta di risorsa deve passare attraverso il banchiere
- Facile da realizzare, ma può richiedere un carico eccessivo:
  - il banchiere può trasformarsi in una strozzatura se invia e riceve molti messaggi
  - Uso poco pratico in un sistema distribuito

Metodo a ordinamento delle marche temporali con diritto di prelazione

- Può gestire ogni situazione di stallo
  - Per semplicità consideriamo il caso di una singola istanza per ciascun tipo di risorsa
- Si assegna ad ogni processo un numero unico di priorità per controllare il diritto di prelazione
  - In base al numero unico si decide se un processo P_i deve attendere un processo P_j (es: P_i attende P_j se ha priorità più alta, altrimenti l'azione di P_i viene annullata)

Metodo a ordinamento delle marche temporali con diritto di prelazione

- Previene lo stallo:
  - Per ogni arco $P_i \rightarrow P_j$ del grafo di attesa, P_i ha priorità più alta di P_j, quindi non può esserci un ciclo
- I processi con priorità molto bassa possono sempre subire l'annullamento delle loro azioni (*attesa indefinita*)
  - Si può evitare usando le marche temporali
  - Ogni processo del sistema riceve una marca temporale unica al momento della sua creazione

Metodo a ordinamento delle marche temporali con diritto di prelazione

- Due metodi complementari di prevenzione che si servono delle marche temporali:
  - Schema **attesa-morte** (wait-die)
  - Schema **ferita-attesa** (wound-wait)

Schema attesa-morte

- Basato su una tecnica senza diritto di prelazione
- Quando P_i richiede una risorsa posseduta da P_j, si permette a P_i di attendere solo se ha una marca temporale TS inferiore a quella di P_j (P_i più vecchio di P_j), altrimenti P_i *muore* (la sua azione viene annullata)

If $TS(P_i) < TS(P_j)$ **then** halt P_i (attendi-wait)
else kill P_i (muori-die)

Schema ferita-attesa

- Basato su una tecnica con diritto di prelazione
- Quando P_i richiede una risorsa posseduta da P_j, si permette a P_i di attendere solo se ha una marca temporale maggiore di quella di P_j (P_i più giovane di P_j), altrimenti si sottrae la risorsa a P_j e la si assegna a P_i (P_j è stato *ferito* da P_i e la sua azione viene annullata)

If $TS(P_i) < TS(P_j)$ **then** kill P_j (ferisci-wound)
else halt P_i (attendi-wait)

Esempio attesa-morte

P_1 più giovane di P_2

P_2 chiede la risorsa che P_1 sta usando
P_2 attende
P_1 rilascia la risorsa
P_2 ottiene la risorsa

P_1 chiede la risorsa che P_2 sta usando
P_1 muore
P_1 viene riavviato e chiede nuovamente la risorsa, che è ancora in possesso di P_2
P_1 muore di nuovo
Ecc..

Esempio ferita-attesa

P_1 più giovane di P_2

P_2 chiede la risorsa che P_1 sta usando
P_1 viene ferito da P_2 e la sua azione viene annullata
P_2 ottiene la risorsa
P_1 viene riavviato

P_1 chiede la risorsa che P_2 sta usando
P_1 attende

attesa-morte VS ferita-attesa

attesa-morte

- Un processo più vecchio deve attendere che uno più giovane rilasci la sua risorsa: più un processo invecchia, più rischia di dover attendere
- Un processo può morire più volte prima di riuscire ad acquisire la risorsa richiesta

ferita-attesa

- Un processo più vecchio non attende mai uno più giovane
- si hanno meno annullamenti

attesa-morte VS ferita-attesa

- Entrambe evitano l'attesa indefinita, purchè quando un processo subisce l'annullamento della sua azione *non* riceva una nuova marca temporale
- Le marche temporali aumentano costantemente
  - Più la marca temporale è grande, più il processo è giovane
  - Un processo la cui azione è stata annullata ha probabilmente la marca temporale minore (più "vecchio") e quindi non sarà annullato ancora una volta
- Si possono verificare annullamenti non necessari

Prevenzione dello stallo

- L'algoritmo di prevenzione può sottrarre risorse anche se non si è verificato alcuno stallo
  - Per prevenire inutili sottrazioni di risorse si può usare un algoritmo di rilevamento delle situazioni di stallo

Esclusione dello stallo

- Tecnica in cui si decide dinamicamente se una richiesta di allocazione di risorsa, nel caso fosse soddisfatta, potrebbe portare ad uno stallo
- In un sistema distribuito non è pratico perché:
  - Ogni nodo deve tener traccia dello stato globale del sistema:
    - sovraccarico di memorizzazione e di comunicazione
  - Il processo di controllare se lo stato globale è sicuro deve essere mutuamente esclusivo:
    - Due nodi potrebbero considerare richieste di processi diversi e giungere concorrentemente alla conclusione che sia sicuro onorare la richiesta, quando in pratica onorandole entrambe si raggiunge uno stallo
  - Controllare la sicurezza dello stallo produce un notevole overhead di elaborazione, nel caso di sistemi distribuiti con un grande numero di processi e risorse

Rilevamento dello stallo

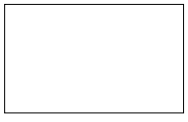
- I processi possono ottenere le risorse che desiderano, se sono libere, e l'esistenza di uno stallo è determinata all'accadere dello stesso
- Si costruisce un **grafo di attesa** che descrive lo stato di assegnazione delle risorse

  Per semplicità consideriamo il caso di un'unica risorsa per ogni tipo, quindi:
- La presenza di un ciclo nel grafo di attesa indica la presenza di uno stallo

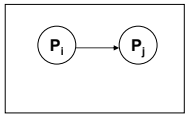
Rilevamento dello stallo

- Il problema in un sistema distribuito è decidere come mantenere il grafo d'attesa:
  - Ogni sito è a conoscenza delle proprie risorse, mentre lo stallo può coinvolgere risorse distribuite
- Bisogna che ogni sito abbia un grafo d'attesa *locale*
  - i nodi del grafo corrispondono a tutti i processi (locali e non) correntemente in possesso di qualche risorsa locale del sito, o che la stanno richiedendo

Costruzione del grafo d'attesa

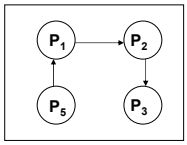


sito S_1

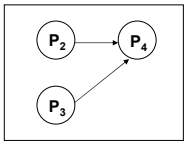


sito S_2

- P_i del sito S_1 ha bisogno di una risorsa posseduta da P_j del sito S_2
- P_i invia un messaggio di richiesta al sito S_2
- Si inserisce l'arco $P_i \longrightarrow P_j$ nel grafo d'attesa locale di S_2



sito S_1



sito S_2

- Se in uno dei grafi d'attesa locali c'è un ciclo, si è verificato uno stallo
  - Se tutti i grafi locali aciclici, non è detto che non vi siano situazioni di stallo
- Per provare che non si è verificato alcuno stallo, bisogna provare che l'unione di tutti i grafi locali è aciclica

Rilevamento dello stallo

- È possibile scegliere tra diversi approcci a seconda che il controllo del sistema sia:
  - **Centralizzato**: un sito è responsabile del rilevamento degli stalli; tutti i messaggi di richiesta e rilascio vengono inviati sia al processo centrale sia al processo che controlla la risorsa
  - **Gerarchico**: i siti sono organizzati in una struttura ad albero; un sito funziona da radice dell'albero; tutti i nodi tranne i nodi foglia collezionano informazioni sull'allocazione di risorse di tutti i nodi sottostanti; un nodo è in grado di rilevare uno stallo se questo coinvolge risorse appartenenti al sottoalbero di cui esso è radice
  - **Distribuito**: tutti i processi cooperano al rilevamento dello stallo

Rilevamento dello stallo

- **Algoritmi centralizzati**:
  - **Pro**: semplici e facili da implementare; il nodo centrale ha informazioni complete e può risolvere lo stallo in modo ottimale
  - **Contro**: overhead notevole di comunicazione (ogni nodo deve mandare informazioni di stato al nodo centrale); vulnerabili al guasto del nodo centrale
- **Algoritmi gerarchici**:
  - **Pro**:non vulnerabili ai guasti di un singolo punto; l'attività di risoluzione degli stalli è limitata se molti degli stalli potenziali sono relativamente localizzati
  - **Contro**: può essere difficile configurare il sistema in modo tale che la maggior parte dei potenziali stalli siano localizzati; altrimenti potrebbe esserci più overhead che nel caso distribuito
- **Algoritmi distribuiti**:
  - **Pro**:non vulnerabili ai guasti di un singolo punto; nessun nodo è caricato dall'attività di rilevamento degli stalli
  - **Contro**: difficili da progettare a causa delle difficoltà di temporizzazione; la risoluzione dello stallo è confusa in quanto diversi nodi possono rilevare lo stesso stallo e non essere a conoscenza di altri nodi coinvolti nello stesso stallo

Metodo centralizzato

- Si costruisce il grafo d'attesa globale unendo tutti i grafi d'attesa locali
- Il grafo globale viene mantenuto da un singolo processo: il **coordinatore per il rilevamento delle situazioni di stallo**
- A causa dei ritardi di comunicazione occorre distinguere due tipi di grafo globale:
  - **Grafo reale**: stato reale, ma sconosciuto, del sistema in qualsiasi istante
  - **Grafo costruito**: approssimazione generata dal coordinatore durante l'esecuzione del suo algoritmo;

Metodo centralizzato

- Il grafo costruito si deve generare in modo che, ogni volta che si invoca l'algoritmo di rilevamento, i risultati indicati siano corretti
- Devono cioè valere le seguenti proprietà:
  - Se esiste uno stallo, la sua esistenza viene correttamente riportata
  - Se si indica uno stallo, il sistema è effettivamente in stallo

- Il grafo d'attesa si può costruire in tre momenti diversi:

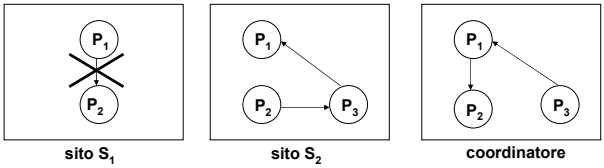
1. Ogni volta che si inserisce o si rimuove un nuovo arco in uno dei grafi d'attesa locali
   - ogni volta che s'inserisce o toglie un'arco da un grafo locale, il sito locale deve inviare un messaggio al coordinatore, che aggiorna il suo grafo globale
2. Periodicamente, quando si verifica un certo numero di cambiamenti nel grafo d'attesa
   - un sito può indicare periodicamente un certo numero di cambiamenti in un unico messaggio
3. Ogni volta che il coordinatore deve invocare l'algoritmo di rilevamento dei cicli

Metodo centralizzato

- Quando si invoca l'algoritmo per il rilevamento delle situazioni di stallo, il coordinatore esamina il grafo globale. Se individua un ciclo, si sceglie una *vittima* da annullare
- Il coordinatore deve informare tutti i siti che un dato processo è stato scelto come vittima, in modo che, a loro volta, ne annullino l'azione
- Si possono compiere annullamenti non necessari o rilevare cicli falsi

Esempio: grafo aggiornato ad ogni cambiamento (punto 1)

- Nel grafo globale possono esistere **cicli falsi**



sito S_1 sito S_2 coordinatore

- P_2 rilascia la risorsa nel sito S_1 e invia il messaggio a
- P_2 richiede una risorsa posseduta da P_3 nel sito S_2 e invia il messaggio b
- Se b arriva prima di a al coordinatore, viene rilevato il falso ciclo $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1$ e avviato il ripristino dello stallo, anche se non si è verificato

Esempio: grafo aggiornato ad ogni cambiamento (punto 1)

- Si possono compiere annullamenti non necessari

sito S_1 sito S_2

- il sito S1 decide di interrompere P2
- Nel frattempo il coordinatore ha rilevato un ciclo e sceglie P3 come vittima
- Vengono annullati sia P2 che P3, anche se sarebbe bastato annullare P2

Algoritmo centralizzato per il rilevamento delle situazioni di stallo

- Il grafo d'attesa viene costruito ogni volta che il coordinatore deve invocare l'algoritmo di rilevamento dei cicli (punto 3)
- Rileva tutte le situazioni di stallo che si verificano effettivamente, mentre non rileva quelle false:
  - Le richieste provenienti da siti diversi sono contraddistinte da identificatori (o marche temporali)
- Quando il processo P_i del sito S_1 richiede una risorsa a P_j del sito S_2, invia un messaggio di richiesta con marca temporale TS
  - L'arco $P_i \rightarrow P_j$ con etichetta TS viene inserito nel grafo locale di S_1
  - L'arco $P_i \rightarrow P_j$ con etichetta Ts viene inserito nel grafo locale di S_2 solo se S_2 ha ricevutó il messaggio di richiesta e non puó concedere immediatamente la risorsa richiesta
- Quando il processo P_i del sito S_1 richiede una risorsa a P_j nello stesso sito, nessuna marca temporale viene associata all'arco $P_i \rightarrow P_j$ e la richiesta si gestisce nel solito modo

L'algoritmo di rilevamento:

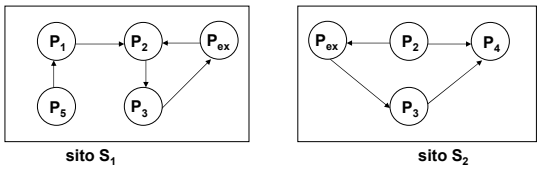
1. Il controllore invia un messaggio d'avvio ad ogni sito del sistema
2. Ricevendo questo messaggio, un sito invia al coordinatore il proprio grafo d'attesa locale. Il grafo riflette uno stato istantaneo del sito, ma non è sincronizzato rispetto ad alcun altro sito
3. Una volta ricevuta una risposta da ogni sito, il controlloe costruisce un grafo in questo modo:
   a) Il grafo costruito contiene un vertice per ogni processo del sistema
   b) Il grafo ha un arco $P_i \rightarrow P_j$ *se e solo se* esiste un arco $P_i \rightarrow P_j$ in uno dei grafi d'attesa, *oppure* un'arco $P_i \rightarrow P_j$ con un'etichetta TS compare in più di un grafo d'attesa

- Se nel grafo costruito esiste un ciclo, il sistema è in stallo
- Se il grafo costruito non contiene alcun ciclo, il sistema non era in stallo quando è stato invocato l'algoritmo di rilevamento (dopo il passo 1)

Metodo totalmente distribuito

- Tutti i controllori condividono in eguale misura la responsabilità del rilevamento delle situazioni di stallo
- Ogni sito costruisce un grafo d'attesa che rappresenta una parte del grafo totale
- Se c'è uno stallo, almeno uno dei grafi parziali contiene un ciclo

Costruzione del grafo d'attesa



sito S_1 sito S_2

- Nel grafo esiste un nodo $P_i \rightarrow P_{ex}$ se P_i attende una risorsa di un altro nodo posseduta da qualche processo
- Nel grafo esiste un arco $P_{ex} \rightarrow P_j$ se un processo in un altro sito attende di acquisire una risorsa attualmente posseduta da P_j in questo sito locale

Algoritmo totalmente distribuito per il rilevamento delle situazioni di stallo

- Se un grafo d'attesa locale contiene un ciclo che non comprende P_{ex},il sistema è in stallo
- Se esiste un ciclo che contiene P_{ex}, c'è la *possibilità* di uno stallo
  - Per accertarsene occorre impiegare un algoritmo distribuito per il rilevamento delle situazioni di stallo
- Si supponga che nel sito S_i il grafo d'attesa locale contenga un ciclo che contiene P_{ex}:

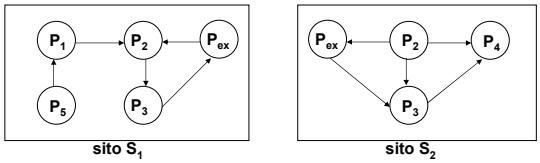
$$P_{ex} \rightarrow P_{k_1} \rightarrow P_{k_2} \rightarrow \ldots \rightarrow P_{k_n} \rightarrow P_{ex}$$

La tranzazione P_k nel sito S_i attende l'acquisizione di una risorsa presente in qualche altro sito, ad esempio S_j

Algoritmo totalmente distribuito per il rilevamento delle situazioni di stallo

- Quando il sito S_i rileva il ciclo, invia a S_j un messaggio di rilevamento di uno stallo contenente le informazioni sul ciclo
- Quando S_j riceve il messaggio, aggiorna il proprio grafo d'attesa locale con le nuove informazioni, quindi cerca un ciclo che non comprenda P_{ex}.
- Se il ciclo esiste, è stato trovato uno stallo
- Se si scopre un ciclo che comprende P_{ex}, S_j trasmette un messaggio di rilevamento al sito interessato, che a sua volta ripete la procedura
- Dopo un numero finito di volte, o si scopre uno stallo o la procedura di rilevamento si arresta

Esempio



- S_1 scopre il ciclo $P_{ex} \rightarrow P_2 \rightarrow P_3 \rightarrow P_{ex}$
- S_1 trasmette a S_2 un messaggio di rilevamento dello stallo
- S_2 aggiorna il grafo con le informazioni ricevute
- S_2 scopre il ciclo $P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2$
- Il sistema è in stallo

- Se S_2 avesse scoperto prima il ciclo nel proprio grafo, e quindi inviato il messaggio, l'esito sarebbe stato lo stesso
- Caso peggiore: S_1 e S_2 scoprono il ciclo nello stesso istante e inviano due messaggi
  - Trasferimento di messaggi non necessario
  - Carico dovuto all'aggiornamento dei due grafi locali
  - Carico dovuto alla ricerca di cicli nei due grafi locali
- Per ridurre il traffico dei messaggi si assegna un identificatore unico $ID(P_i)$ ad ogni processo P_i
  - Quando il sito S_k scopre un ciclo $P_{ex} \rightarrow P_{k_1} \rightarrow P_{k_2} \rightarrow \ldots \rightarrow P_{k_n} \rightarrow P_{ex}$ invia un messaggio di rilevamento ad un altro sito se $ID(P_{k_n}) < ID(P_{k_1})$
  - Altrimenti continua la sua normale esecuzione, lasciando il compito di avviare l'algoritmo a qualche altro sito
  - (nell'esempio precedente, è il sito S_2 ad avviare l'algoritmo)

Algoritmi di elezione

- Molti algoritmi distribuiti impiegano un processo coordinatore che esegue funzioni richieste da altri processi del sistema
- Se il processo coordinatore si guasta a causa di un difetto del sito in cui risiede, il sistema può continuare l'esecuzione solo avviando una nuova copia del coordinatore in un altro sito
- Gli algoritmi che determinano dove si debba avviare una nuova copia del coordinatore si chiamano

Algoritmi di elezione

- Si fondano sul presupposto che a ogni processo attivo del sistema sia associato un numero di priorità unico
  - Supponiamo che il numero di priorità del processo P_i sia *i*
- Il coordinatore è sempre il processo col numero di priorità maggiore
  - Quando il coordinatore si guasta, l'algoritmo deve eleggere il processo attivo col numero di priorità maggiore
  - Questo numero deve essere inviato ad ogni processo attivo del sistema
- Devono fornire un meccanismo per consentire ai processi ripristinati dopo un guasto di identificare l'attuale coordinatore
  - Per semplicità supponiamo che un processo che si è guastato riconosca, al momento del ripristino, di essersi guastato e intraprenda le opportune azioni per ricongiungersi all'insieme dei processi attivi

Algoritmo dello spaccone

- Si applica in sistemi in cui ogni processo può inviare un messaggio ad ogni altro processo del sistema
- Richiede n^2 messaggi per un'elezione (n numero dei processi nel sistema)
- Supponiamo che P_i invii una richiesta al coordinatore e non riceva risposta entro un intervallo *t*
  - Si suppone che il coordinatore sia guasto e P_i tenta di eleggersi nuovo coordinatore

Algoritmo dello spaccone

- P_i invia un messaggio di elezione ad ogni processo con numero di priorità maggiore
- se entro l'intervallo *t* non giunge risposta, P_i suppone che tutti i processi con numeri maggiori di *i* siano guasti e si elegge nuovo coordinatore
  - P_i avvia una nuova copia del coordinatore e invia un messaggio per informare tutti i processi con numero $<i$ che P_i è il nuovo coordinatore
- Se arriva una risposta, P_j attende per un intervallo *t'* un messaggio che lo informi dell'elezione di uno tra i processi con numero maggiore
  - Qualche altro processo si sta eleggendo coordinatore e deve indicare il risultato entro l'intervallo *t'*
  - se entro l'intervallo *t'* non viene inviato alcun messaggio, si suppone che il processo con priorità maggiore sia guasto e P_i deve riavviare l'algoritmo

Algoritmo dello spaccone

- Se P_i non è il coordinatore, in un momento qualsiasi dell'esecuzione può ricevere dal processo P_j uno tra i seguenti messaggi:
  1. P_j è il nuovo coordinatore (j>i); P_i registra questa informazione
  2. P_j ha avviato un'elezione (j<i); P_i invia una risposta a P_j e inizia il proprio algoritmo di elezione, purchè non lo abbia già iniziato
- Il processo che completa l'algoritmo ha il numero maggiore e viene eletto coordinatore
- Dopo che un processo guasto è stato ripristinato comincia l'esecuzione dell'algoritmo
  - Se non ci sono processi attivi con numero maggiore, s'impone come coordinatore anche se ne esiste uno attualmente attivo con numero minore (da qui il nome)

Algoritmo ad anello

- Si applica a sistemi organizzati a forma d'anello:
  - I collegamenti sono *unidirezionali*
  - I processi inviano i messaggi ai loro vicini di destra
- Richiede n^2 messaggi per un'elezione (n numero dei processi nel sistema)
- Usa una struttura dati detta lista attiva:
  - contiene i numeri di priorità di tutti i processi attivi nel sistema al termine dell'algoritmo
  - Ogni processo conserva la propria lista attiva

Algoritmo ad anello

- Se P_i rileva un guasto nel coordinatore:
  - crea una nuova lista attiva inizialmente vuota
  - invia un messaggio *elect(i)* al suo vicino destro
  - aggiunge il numero *i* alla propria lista attiva
- Se riceve un messaggio *elect(j)* dal vicino sinistro, deve rispondere in uno dei tre modi:
  a) Se è il primo messaggio *elect* che ha ricevuto o inviato:
     - ✓ Crea una nuova lista attiva con i numeri *i* e *j*
     - ✓ Invia il messaggio *elect(i)* seguito dal messaggio *elect(j)*
  b) Se $i \neq j$ (il messaggio non contiene il numero *i*):
     - ✓ P_i aggiunge *j* alla propria lista attiva
     - ✓ Inoltra il messaggio al vicino destro
  c) Se $i = j$ (P_i riceve *elect(i)*):
     - ✓ La lista attiva di P_i contiene tutti i processi attivi del sistema
     - ✓ P_i può determinare il nuovo coordinatore (numero maggiore nella lista)

Algoritmo ad anello

- Non specifica come un processo nella fase di ripristino possa determinare il numero del processo attualmente coordinatore
- Soluzione:
  - Il processo nella fase di ripristino invia un messaggio d'interrogazione
  - Il messaggio viene ritrasmesso nell'anello fino al coordinatore, che invia una risposta contenente il proprio numero

Raggiungimento di un accordo

- Affinché un sistema sia affidabile, deve esistere un meccanismo che permetta a un insieme di processi di accordarsi su un valore comune
- L'accordo può non essere raggiunto per diversi motivi:
  - Comunicazione inaffidabile del calcolatore (perdite o alterazioni dei messaggi)
  - Processi difettosi (comportamento imprevedibile, tentativi di distruggere l'integrità del sistema)

Comunicazione inaffidabile

- Supponiamo che:
  - se dei processi si guastano, lo facciano in modo chiaro
  - il mezzo di comunicazione sia *inaffidabile*
- Supponiamo che P_i del sito S_1 abbia inviato un messaggio a P_j del sito S_2, e debba sapere se P_j ha ricevuto il messaggio per decidere come procedere con l'elaborazione
  - Es.: if (P_j ha ricevuto il messaggio) calcola f
    else calcola g

Comunicazione inaffidabile

- Per rilevare i guasti si può usare uno schema d'attesa:
  - Quando P_i invia un messaggio, specifica anche un'intervallo di tempo entro il quale è disposto ad attendere un messaggio di conferma da parte di P_j
    - Se P_i riceve la conferma entro l'intervallo specificato, può concludere con certezza che P_j ha ricevuto il messaggio
    - Altrimenti P_i deve ritrasmettere il messaggio e attendere una conferma
  - Quando P_j riceve il messaggio invia subito un messaggio di conferma a P_i
  - Questa procedura continua finchè:
    - P_i ottiene il messaggio di risposta (P_i calcola f), oppure
    - il sistema gli comunica che il sito S_2 è fuori servizio (P_i calcola g)
      - Se queste sono le uniche possibilità, P_i deve attendere finchè non riceve la comunicazione che si è verificata una delle due situazioni

Comunicazione inaffidabile

- Supponiamo che anche P_i abbia bisogno di sapere se P_i ha ricevuto il messaggio di conferma
  - Es.: calcola f solo se P_i ha ricevuto la conferma
- P_j e P_j calcolano f solo se entrambi hanno raggiunto un *accordo*
- Alla presenza di un guasto non è possibile svolgere questo compito
  - **non è possibile che in un ambiente distribuito i processi P_j e P_j concordino completamente sui rispettivi stati**
- I processi non possono mai avere la certezza che entrambi eseguiranno il calcolo di f

Processi difettosi

- Supponiamo che:
  - il mezzo di comunicazione sia affidabile
  - i processi si guastino in in modi imprendibili
- Consideriamo un sistema di *n* processi, dei quali non più di *m* siano difettosi
- Supponiamo che ogni processo P_i abbia un numero privato di V_i
- Vogliamo costruire un algoritmo che permetta ad ogni processo di costruire un vettore $X_i = (A_{i,1}, A_{i,2}, \ldots, A_{i,n})$ tale che valgano le seguenti condizioni:
  1. Se P_j è un processo *non difettoso*, allora $A_{i,j} = V_j$
  2. Se P_i e P_j sono *entrambi non difettosi*, allora $X_i = Xj$

Processi difettosi

- Le diverse soluzioni hanno in comune le seguenti proprietà:
  1. Si può ideare un algoritmo corretto solo se $n \geq 3 \times m + 1$
  2. Il ritardo maggiore nel raggiungimento dell'accordo è proporzionale al ritardo dovuto a m + 1 scambi di messaggi
  3. Il numero dei messaggi richiesti per raggiungere un accordo è elevato; nessun singolo processo è affidabile, quindi tutti i processi devono raccogliere tutte le informazioni e prendere le proprie decisioni
- Soluzione generale complicata

Esempio di algoritmo

- Se m = 1 e n = 4, l'algoritmo richiede due giri di scambio di informazioni:
  1. Ogni processo invia il proprio valore privato agli altri tre processi
  2. Ogni processo invia le informazioni ottenute nel primo giro a tutti gli altri processi
     - un processo difettoso può rifiutare di inviare messaggi; un processo non difettoso può scegliere un valore arbitrario e assumere che sia stato inviato dal processo difettoso
- Ora ogni processo P_i *non difettoso* può costruire il proprio vettore:
  1. $A_{i,i} = V_i$
  2. Per $j \neq i$:
     - se almeno due dei tre valori registrati per il processo P_j durante lo scambio d'informazioni coincidono, impostare il valore di $A_{i,j}$ con il valore di maggioranza
     - altrimenti impostarlo con un valore predefinito, ad esempio *nil*

Stallings, cap.14 ; Silberschatz, cap.17