Le Azioni Atomiche

AZIONI ATOMICHE

Strumento di più alto livello per la strutturazione di programmi concorrenti

Applicazioni:

- sistemi operativi distribuiti
- applicazioni transazionali

- Utilizzo nella costruzione di programmi **tolleranti vari tipi di malfunzionamenti**
- E' un'astrazione realizzata con i meccanismi linguistici offerti dal linguaggio concorrente, piuttosto che un nuovo costrutto linguistico

CONSISTENZA DEI DATI

- Ogni oggetto astratto può trovarsi in stati **consistenti** o **inconsistenti** a seconda che si sia verificata o meno una particolare relazione (**invariante del tipo**) fra i valori delle variabili componenti l'oggetto
- Ogni tipo ha una sua **relazione invariante,** che lo caratterizza dal punto di vista semantico
- Ogni operazione del tipo **deve** essere programmata in modo da lasciare l'oggetto su cui opera **in uno stato consistente (**cioe` in uno stato in cui l'invariante sia soddisfatta**)**

- Durante l'esecuzione dell'operazione l'oggetto può passare attraverso stati inconsistenti, che tuttavia **non devono essere visibili** ad altre operazioni

- Modello ad ambiente globale: **monitor**
- Modello ad ambiente locale: **processo servitore**

CONSISTENZA DEI DATI E PARALLELISMO

Sia **O = [O1, O2,….. On]** l'insieme degli oggetti su cui operano più programmi concorrenti

➔ necessità di mantenere la consistenza **dell'insieme di oggetti** al termine dell'esecuzione di un programma

- **La relazione di consistenza R** (**invariante)** è **relativa** all'insieme di oggetti

ESEMPIO: *applicazioni bancarie*

  - **oggetti**: conti correnti
  - **programmi**: operazioni di lettura, modifica, ecc., che riguardano più oggetti
- Ad esempio, Operazione di trasferimento: dati gli oggetti O1e O2, *spostare* **x** *da* O1 *a* O2.

➔ L'invariante e`: R = *valore* O1 + *valore* O2 = *costante*

- Durante l'esecuzione del programma, quando **x** è stata tolta da O_1 ma non ancora sommata a O_2, si ha **stato complessivo inconsistente** (anche se i due oggetti si trovano singolarmente in stato consistente)
- La mutua esclusione sui singoli oggetti garantisce l'atomicità delle operazioni su di essi, ma non **dell'intera operazione su tutto l'insieme**
- E' necessario che l'intero programma che interessa gli oggetti possa essere considerato **atomico**, cioè non divisibile (vale per entrambi i modelli).

Soluzioni

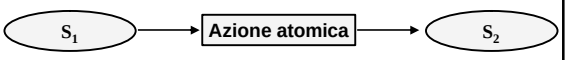
1. **nuova astrazione** che racchiude tutti gli oggetti e tutte le operazioni possibili (monitor o processo servitore)

   ➔ ***Limiti:***
   - Le operazioni possono **non essere note** al momento delle definizione dell'astrazione
   - non e` possibile operare direttamente sugli oggetti

2. Associare ad ogni oggetto un gestore ed obbligare i processi applicativi a **richiedere** l'uso dedicato **di tutti gli oggetti** e **rilasciarli** solo dopo che il nuovo stato **consistente** è stato raggiunto.

Consistenza dei Dati e Malfunzionamento

- Fino ad ora abbiamo considerato l'azione **atomica** come un'operazione che in forma ***non divisibile*** fa transitare un insieme di oggetti da uno stato consistente iniziale (S_1) ad un altro consistente finale (S_2).



- Durante l'esecuzione dell'azione atomica si può verificare una qualche **condizione** anomala che ne impedisce il completamento.
- Affinché gli oggetti non rimangano in uno stato **inconsistente**, è necessario un meccanismo di ***recupero*** che riporti gli oggetti in uno stato **consistente.**

Proprieta` "*TUTTO O NIENTE*"

Indicando con S' lo stato risultante dall'esecuzione dell'azione atomica, **si deve avere:**

S'= S1 **oppure** S'=S2

dove S1 e S2 sono rispettivamente lo stato iniziale e finale.

La proprietà indicata prende il nome di

tutto o niente

Proprietà tutto o niente (segue)

- Un'azione atomica che gode della proprietà del tutto o niente è in grado di tollerare il *verificarsi di condizioni anomale* durante la sua esecuzione, senza lasciare gli oggetti in stato inconsistente.

Esempio dell'applicazione bancaria:

- Evento anomalo dopo che x è stata addebitata ad O1, ma non ancora accreditata ad O2.
- O1 e O2 sono globalmente inconsistenti.

➔ Necessità di un meccanismo di recupero che riporti lo stato complessivo in uno stato consistente.

Azioni Atomiche
Proprieta` Fondamentali

- Per garantire la **consistenza dei dati**, l'azione atomica deve possedere due proprietà fondamentali:
  - ***Tutto o niente*** (o atomicità nei confronti di eventi anomali)
  - ***Serializzabilità*** (o atomicità nei confronti della concorrenza)

Serializzabilita`

- La proprietà di serializzabilità delle azioni atomiche è simile a quella di indivisibilità delle azioni primitive:

  Assicura che ogni azione atomica operi <u>sempre</u> su un insieme di oggetti il cui stato iniziale è consistente *ed* ***i cui stati parziali, durante l'esecuzione, non sono visibili ad altre azioni <u>concorrenti</u>***

- Sia **A** un'azione atomica che opera su un insieme di oggetti **O= {O1, O2,….. On}**
- Siano **Richiesta** (**Oi)** e **Rilascio** (**Oj)** le operazioni per richiedere al gestore di ogni oggetto l'uso esclusivo dell'oggetto
- La serializzabilità viene garantita allocando dinamicamente singoli oggetti **in modo dedicato** alle azioni atomiche secondo il seguente **protocollo**:
  a. Ogni oggetto deve essere **acquisito** da A in modo esclusivo prima di qualunque azione su di esso:
     ➔ Richiesta (**Oi**) e` bloccante se l'oggetto **Oi** non è disponibile
  b. Nessun oggetto deve essere **rilasciato** prima che siano eseguite tutte le operazioni su di esso;
  c. Nessun oggetto può essere **richiesto** dopo che è stato effettuato un **rilascio** di un altro oggetto

ESEMPIO:

```
A1 :  {  X= X+20;
         Y= Y+20;
      }
```

```
A2 : { X= X*20;
       Y= Y*20;
     }
```

Sia **X = Y** la relazione di consistenza:

- Ogni azione atomica preserva la consistenza e quindi anche una qualunque **esecuzione sequenziale** di $\mathbf{A_1}$ e $\mathbf{A_2}$
- Se A_1 e A_2 sono eseguite **in parallelo** possono nascere inconsistenze

Rispettando i due primi requisiti (a e b) si può avere

```
A1 :{   Richiesta(X);
        X= X+20;
        Rilascio(X);
        Richiesta(Y);
        Y= Y+20;
        Rilascio(Y);
     }
```

```
A2:{  Richiesta(X);
      X= X*10;
      Rilascio(X);
      Richiesta(Y);
      Y= Y*10;
      Rilascio(Y);
      }
```

IPOTESI: dopo Rilascio(X) ma prima di Richiesta(Y) da parte di $\mathbf{A_1}$, viene eseguita $\mathbf{A_2}$

X_i Y_i valori iniziali di X e Y

X_f Y_f valori finali di X e Y

X_i=Y_i per ipotesi

$X_f = (X_i + 20) * 10$

$Y_f = (Y_i * 10) + 20$

E quindi

X_f *diverso* Y_f

- L'inconsistenza è dovuta al non rispetto del requisito c)
- Il requisito c) assicura che quando **un'azione rilascia un oggetto contenente il valore finale**, nessuno degli altri oggetti su cui l'azione opera è libero e contenente il valore iniziale

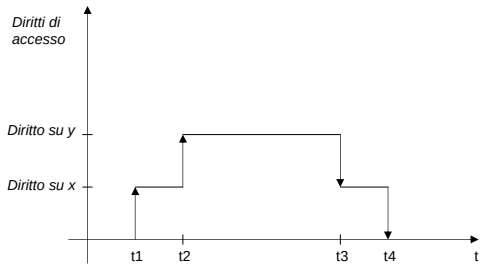
TWO PHASE LOCK PROTOCOL

I° fase (fase crescente)

l'azione atomica acquisisce gli oggetti ed opera su di essi

II° fase (fase calante)

inizia non appena viene eseguito il primo rilascio e durante essa non possono essere acquisiti ulteriori oggetti




```
A1 :   {                          A2 :{
         Richiesta(X);                    Richiesta(X);
         X= X+20;                         X= X*10;
         Richiesta(Y);                    Richiesta(Y);
         Rilascio(X);                     Rilascio(X);
         Y= Y+20;                         Y= Y*10;
         Rilascio(Y);                     Rilascio(Y);
       }                                  }
```

•Si può verificare che le inconsistenze esaminate precedentemente non possono più verificarsi qualunque sia il grado di concorrenza tra le due azioni atomiche

•E` così soddisfatta la **condizione di serializzabilità.**

Per soddisfare la condizione del "**tutto o niente**" occorre definire un ulteriore requisito:

d. Nessun oggetto può essere rilasciato prima che l'azione atomica **abbia completato la sua esecuzione**

In altri termini, i rilasci devono costituire le ultime operazioni dell'azione atomica

ESEMPIO: **A1** e **A2** soddisfano a), b), c) ma non d).

A1 e A2 con riferimento all'esempio precedente

1. **A_1** in esecuzione:
   - viene acquisito X e viene eseguita X = X+20
   - viene acquisito Y
   - viene rilasciato X, contenente il valore finale
2. **A_2** in esecuzione:
   - viene acquisito X e viene eseguita X = X*10
3. A causa di un **malfunzionamento** (es. condizione di stallo in cui il processo che esegue **A_1** è coinvolto) l'oggetto **X** viene riportato al valore che aveva **prima** che iniziasse **A_1** ;
4. Gli effetti dell'ultima operazione di **A_2** su X vengono distrutti. E' necessario pertanto ripristinare lo stato iniziale degli oggetti su cui opera **A_2** .

L'operazione di distruzione degli effetti di un'azione atomica ed il ripristino del valore iniziale degli oggetti viene indicato con il termine di ***aborto.***

Effetto *domino*: l'aborto di un'azione atomica genera, **come effetto collaterale**, l'aborto di una diversa azione atomica (e così via)

Causa: l'azione atomica rilascia un oggetto **prima di aver completato** la sequenza di operazioni su tutti gli oggetti interessati e di aver raggiunto uno **stadio di avanzamento** tale da **garantire** che, da quel punto in poi, qualunque evento anomalo accada, l'azione atomica **non sarà più abortita.**

Operazione commit

- Per garantire la proprietà ***tutto o niente*** (come si vedrà) è necessario che il meccanismo di recupero da malfunzionamento si comporti in **maniera diversa a seconda** dell'istante in cui l'evento anomalo si verifica. Dovrà:
  - **abortire l'azione** se il malfunzionamento avviene quando gli oggetti sono in stato inconsistente (effetto ***niente***)
  - **garantire il completamento** dell'azione quando gli oggetti sono nel nuovo stato consistente (***effetto tutto***)
- L'operazione primitiva ***commit*** discrimina i due tipi di comportamento del meccanismo di recupero da malfunzionamento;
- Viene eseguita quando tutti gli oggetti sono al valore finale e produce come effetto l'impossibilità di aborto dell'azione atomica (***completamento con successo***).

•Il requisito d) si può riformulare nel seguente modo:

Ogni azione atomica deve rilasciare i propri oggetti **dopo** l'operazione di completamento con successo (***commit***)

```
A1 :  {                          A2 :  {
        Richiesta(X);                    Richiesta(X);
        Richiesta(Y);                    Richiesta(Y);
        X:= X+20;                        X:= X*10;
        Y:= Y+20;                        Y:= Y*10;
        commit;                          commit;
        Rilascio(X);                     Rilascio(X);
        Rilascio(Y);                     Rilascio(Y);
      }                                }
```

➔ Rimane soddisfatto il **two phase lock protocol**

Le azioni atomiche vengono spesso utilizzate come strumento per strutturare applicazioni transazionali, ad esempio gestione di basi di dati

Richiesta (x) viene indicata con lock (x)

Rilascio (x) viene indicata con unlock (x)

•**Shared lock** per acquisire un oggetto su cui operare in sola lettura

•**Exclusive lock** lettura e scrittura

TRANSAZIONE ⇒ sequenza di operazioni effettuate su data base che fanno passare il sistema da uno stato all'altro, ambedue consistenti

Deve rispettare 4 proprietà:

ACID (Atomic, Consistency, Isolation, Durability)

1. ATOMICITA' ⇒ transazione come **entità indivisibile**
2. CONSISTENZA ⇒ evoluzione del data base da uno stato corretto all'altro
3. ISOLAMENTO ⇒ informazioni **protette** durante l'esecuzione delle transazione
4. DURATA ⇒ una volta che la transazione sia completata i suoi effetti sul data base hanno carattere **duraturo**. Possono essere alterati soltanto da altre transazioni e non da **malfunzionamenti** del sistema.

Proprietà "tutto o niente"

- Azione atomica termina in uno dei **due modi**:

  Terminazione normale - l'azione atomica **completa** l'intera sequenza delle operazioni sugli oggetti. Nuovo stato consistente

  Terminazione anomala – l'azione atomica **non completa** l'intera sequenza di operazioni sugli oggetti che devono essere ripristinati al **valore iniziale** (azione atomica abortita)

- La sequenza dei rilasci fa parte dell'operazione di completamento con successo o di aborto

Cause di terminazione anomala:

1. Il verificarsi di un'eccezione sollevata durante una delle operazioni sugli oggetti. Il **processo** esegue la primitiva ***abort***
2. Il verificarsi di un **malfunzionamento** hardware o di una condizione di blocco critico. Il sistema di recupero da malfunzionamento forza l'aborto dell'azione atomica (stato iniziale)

- In entrambi i casi è necessario un meccanismo di supporto alle azioni atomiche in grado di ripristinare, per ogni oggetto, il suo stato iniziale.

- Il meccanismo necessita di informazioni relative allo stato corrente delle operazioni ed allo stato iniziale degli oggetti.
- Nel caso di eccezioni durante l' esecuzione dei programmi (caso 1) queste informazioni sono facilmente disponibili
- Nel caso di malfunzionamento hardware (interruzione di energia elettrica, guasto fisico, ecc.) si ha la perdita di tutte le informazioni residenti in RAM

IPOTESI: Le informazioni su memoria di massa rimangono **inalterate**. Le informazioni necessarie per il recupero vengono mantenute in memoria di massa.

Memoria stabile

- Esiste la possibilità che le informazioni contenute nella memoria di massa risultino alterate (non consistenti) in seguito ad un malfunzionamento

  Es: Caduta (*crash*) dell'elaboratore mentre sta trasferendo informazioni nella memoria di massa

Memoria stabile: astrazione, con la proprietà di contenere le informazioni necessarie al recupero e di non essere soggetta ad alcun tipo di malfunzionamento

- Realizzata tramite uso di **ridondanza** delle informazioni
- Le operazioni per leggere e scrivere su questa memoria sono **atomiche** (*stable-read, stable-write*). Un malfunzionamento che si verifichi durante la loro esecuzione deve avere gli stessi effetti di un malfunzionamento che si sia verificato **o prima o dopo** l'esecuzione stessa (tutto o niente)
- Per specificare il **tipo d operazione** che deve eseguire il meccanismo di recupero dopo la caduta dell'elaboratore è necessario caratterizzare lo stato del processo in base allo stato di avanzamento nell'esecuzione dell'azione atomica

STATI DI UN PROCESSO

Stato working

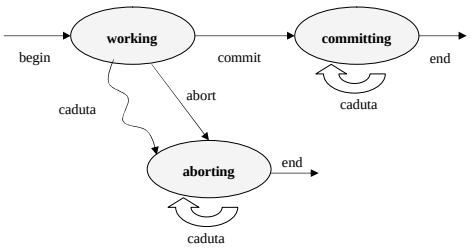
- Durante l'esecuzione del corpo dell'azione atomica. Quando il processo è in questo stato gli oggetti sono *inconsistenti*.
- Se l'elaboratore cade, il meccanismo di recupero deve *abortire* l'azione atomica.

Stato committing

- Durante la terminazione corretta dell'azione. Gli oggetti sono al loro *stato finale*. Il processo commuta in tale stato tramite la *commit*. Se l'elaboratore cade il meccanismo di recupero deve *completare* l'azione (valori finali già disponibili)

Stato aborting

- Durante la terminazione anomala dell'azione atomica. Gli oggetti devono essere ripristinati al loro *valore iniziale*.Il processo commuta in tale stato tramite *abor*t. Se l'elaboratore cade durante l'azione di aborto, il meccanismo di recupero deve garantire il *completamento del ripristino dei valori iniziali degli oggetti*



•Le informazioni sullo stato in cui si trova un processo che esegue un'azione atomica vengono tenute aggiornate in una **struttura dati** (DESCRITTORE DI AZIONE) allocata in memoria stabile

•Primitiva **begin action** crea in memoria stabile un descrittore di azione , inizializzando lo stato del processo a "working"

•Le primitive **abort** e **commit** dovranno commutare atomicamente lo stato del processo ad "aborting" e "committing"

⇓

•Durante la riattivazione dell'elaboratore, il **meccanismo di recupero** può desumere, analizzando il descrittore dell'azione atomica, lo stato dei processi al momento del malfunzionamento

OPERAZIONE DI ABORT

- Gli oggetti risiedono in memoria stabile. All'inizio dell'azione atomica viene creata una copia degli oggetti (copia di lasvoro) in memoria volatile sulla quale eseguire le operazioni
- L'aborto dell'azione atomica coincide con la distruzione delle copie di lavoro in memoria volatile (in memoria stabile esistono i valori iniziali)
- Soltanto se l'azione termina correttamente i valori finali delle copie di lavoro sono salvati nella copia valida in memoria stabile

Schema di programma di un'azione atomica

< creazione del descrittore dell'azione atomica>;
< richiesta esclusiva degli oggetti residenti in memoria stabile>;
/*fase crescente del two phase lock protocol*/
<creazione copie volatili degli oggetti>;
<sequenza di operazioni sulle copie volatili>;
/*corpo dell'azione atomica*/
if (<assenza di malfunzionamenti>)
<terminazione (descrittore)>;
else *abort* (descrittore);
<distruzione copie volatili>;
<rilascio oggetti in memoria stabile>;
/*fase decrescente del two phase lock protocol*/
<eliminazione descrittore>

•La procedura Terminazione viene eseguita, nel caso di corretto completamento dell'azione atomica, per salvare in memoria stabile i valori finali degli oggetti.

•La procedura prevede una sequenza di operazioni *stable write*; se l'elaboratore cade a metà sequenza si ha uno *stato inconsistente*.

Copia delle intenzioni: Copia dei valori finali creata in memoria stabile prima della **commit** (*stato working*), senza modificare la copia stabile originale.

Solo se la copia delle intenzioni è stata correttamente memorizzata, viene successivamente trasferito il suo valore nella copia originale.

<creazione copia intenzioni in memoria stabile>
commit
<trasferimento valori da copia intenzioni a copia oggetti in memoria stabile>
<distruzione copia intenzioni>

•Le due fasi della procedura sono intervallate dalla primitiva **commit**.

•Se l'elaboratore cade durante la prima fase, la copia originale degli oggetti in memoria stabile rimane inalterata al valore iniziale.L'azione viene abortita e viene distrutta la copia delle intenzioni.

•Se la caduta avviene nella seconda fase, la copia finale viene corrotta ma rimane valida in memoria stabile la copia delle intenzioni.

•In fase di riattivazione la seconda fase della procedura viene eseguita dall'inizio (fase committing).

Realizzazione della memoria stabile

- Memoria stabile (astrazione) è un tipo di memoria con le seguenti proprietà:
  - non è soggetta a malfunzionamenti
  - le informazioni in essa residenti non vengono perdute o alterate a causa della caduta dell'elaboratore.
- La prima proprietà si ottiene usando tecniche che fanno uso di **ridondanza.**
- La seconda proprietà richiede che le operazioni di lettura e scrittura siano delle operazioni atomiche (proprietà del *tutto o niente*). **Stable read e stable write.**

Tipi di malfunzionamento

Si prendono in considerazione i seguenti tipi di errore:

a) **errori in lettura**. Possono essere eliminati rileggendo le informazioni desiderate. Uso dei codici di ridondanza ciclica (CRC).
(errori transitori)

b) **errori in scrittura**. Sono rilevabili rileggendo le informazioni scritte e confrontandole con quelle originali. L'errore può essere eliminato riscrivendo le informazioni (errori transitori).

c) **alterazioni delle informazioni** a causa di disturbi o di guasti hardware che perdurano per un certo numero n di letture consecutive
(errori persistenti).

Per i problemi del tipo c) si fa ricorso alla tecnica delle *copie multiple*.
Ridondanza di livello due: duplicazione di tutte le informazioni su due unità a disco distinte.

Classificazione dei malfunzionamenti

- Malfunzionamenti dovuti a **disturbi temporanei**: gli esempi a) e b)
- Malfunzionamenti dovuti a **guasti hardware**: esempio c) ed **errori prodotti dalla caduta dell'elaboratore durante una scrittura.**

Disco permanente: astrazione ottenuta eliminando ogni tipo di guasto temporaneo.

- Utilizzo di **una coppia di dischi permanenti** per realizzare l'astrazione memoria stabile.

- Un **blocco stabile** è costituito da una coppia di blocchi uno per ogni disco permanente.
- ***Ipotesi di base*:** le due copie non vengono alterate entrambe dallo stesso malfunzionamento. In tal modo, in fase di lettura almeno una deve contenere il valore corretto.Il valore contenuto in un blocco stabile coincide con il valore del primo blocco se questo contiene dati corretti, altrimenti coincide con il valore del secondo blocco.
- **Atomicità di stable read e stable write** .

Proprietà di serializzabilità. La memoria stabile è realizzata come un **monitor** il che garantisce l'indivisibilità delle sue operazioni.

Proprietà "tutto o niente"

Stable read gode della proprietà del tutto o niente perché non effettua alcuna modifica.

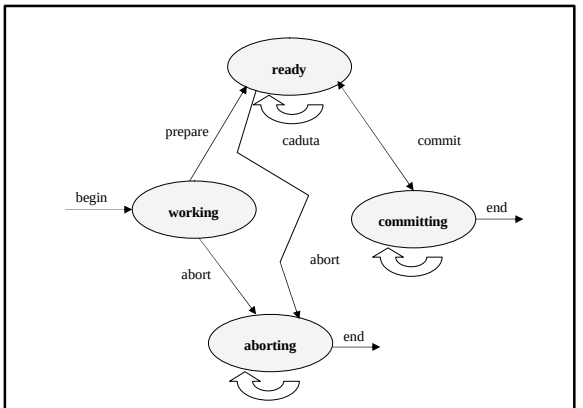
Stable write. Se si verifica un guasto durante la operazione di scrittura:

- •sul primo blocco, questo viene alterato, ma rimane intatto il secondo (*effetto niente* della operazione).
- •sul secondo blocco, rimane valido il primo (*effetto tutto* dell'operazione)
- •tra le due operazioni: il risultato è *non consistente*. Il meccanismo di recupero legge i due blocchi:
  - •Se uno è alterato viene sostituito con l'altro.
  - •Se sono entrambi corretti, ma con valori diversi significa che c'è stato un guasto tra le due scritture
  - ➔ Si copia il primo blocco nel secondo.

AZIONI ATOMICHE MULTIPROCESSO

- In alcune applicazioni può risultare conveniente che le operazioni sugli oggetti di una azione atomica siano eseguite non da un solo processo, ma da **più processi**.
- Le azioni **atomiche multiprocesso** sono tipiche dei sistemi distribuiti dove gli oggetti possono essere allocati su nodi diversi.
- L'elaborazione complessiva rimane comunque un'azione atomica se gode delle due proprietà di **serializzabilità** e **del tutto o niente**.
- L'ipotesi adottata è che i singoli processi operino ciascuno su **oggetti diversi**; Ciò consente di evitare condizioni di *blocco critico*.

- Se ogni processo segue il **two phase lock protocol**, rilasciando gli oggetti su cui opera dopo la terminazione dell'azione, è assicurata la proprietà di **serializzabilità** dell'azione atomica.
- Per soddisfare la proprietà del **tutto o niente** è necessario che tutti i processi completino le loro operazioni con lo stesso risultato: o con successo o con *abort.*
- Un comportamento difforme (anche di un solo processo partecipante) porterebbe, alla fine dell'azione, alcuni oggetti al valore finale, altri al valore iniziale (*stato inconsistente*).
- Necessità di introdurre un nuovo stato, stato **ready.**
- Caratterizza un processo quando ha completato la propria sequenza di azioni ed è pronto a terminare con successo (*commit*), ma deve attendere che gli altri processi completino le loro operazioni.



- La transazione da *working* a *ready* si ha quando il processo esegue la primitiva **prepare**. Con essa il processo perde il diritto di abortire unilateralmente. Nello stato *ready* diventa disponibile a terminare con successo o ad abortire.
- Il protocollo che ciascun processo esegue per negoziare il tipo di completamento prende il nome di **two phase commit protocol**.
- Nella *prima fase* ciascun processo specifica la propria opzione (completamento con successo o aborto).
- Nella *seconda fase* viene verificata l'opzione degli altri: se tutti hanno optato per terminazione con successo, tutti transitano in stato *committing*; se almeno uno ha abortito, tutti transitano in stato *aborting*.
- La primitiva **prepare** separa le due fasi del protocollo.

- Se cade l'elaboratore quando un processo è in stato *ready*, il meccanismo di recupero deve ripristinare per tale processo lo stato *ready*.
- Infatti il processo non può essere riattivato in stato *aborting* poichè con *prepare* ha perduto il diritto di abortire unilateralmente, né in stato *committing* perché ciò presuppone che tutti siano pronti a terminare correttamente.
- Il descrittore di azione atomica in memoria stabile dovrà tenere aggiornato lo stato di tutti i processi partecipanti.
- La realizzazione dell'azione multiprocesso cambia a seconda del *modello di interazione* tra processi.
  - Nel modello a *memoria comune* il descrittore di azione è un *monitor*.
  - Nel modello a *scambio di messaggi* il descrittore è un oggetto privato di un processo detto *coordinatore dell'azione atomica*

Modello a memoria comune

- L'azione atomica viene iniziata da un processo che ne crea in memoria stabile il descrittore e attiva in parallelo un certo numero di processi che la eseguono.
- Quando tutti i processi hanno completato le loro operazioni, con successo o terminando con aborto, il processo iniziale riprende il controllo e termina l'azione cancellando dalla memoria stabile il descrittore.

Schema del funzionamento del processo i-esimo:

- Richiesta esclusiva degli oggetti
- Creazione delle copie volatili degli oggetti
- Sequenza di operazioni sulle copie volatili
- Terminazione:
  - *con aborto*
    - Entra nello stato *aborting*. Lo stato viene riportato sul descrittore della periferica. Vengono risvegliati i processi nello stato di *ready*.
  - *con successo*
    - Crea la copia delle intenzioni in memoria stabile
    - Transita nello stato di *ready* attraverso l'esecuzione della *prepare; v*iene modificato il suo stato da working a *ready* nel descrittore dell'azione atomica.
    - Verifica se almeno un processo ha abortito (è in stato aborting). In caso affermativo, distrugge la copia delle intenzioni ed esegue abort

(segue *Terminazione con successo*)

In caso *negativo* si possono verificare due casi:

- Qualche altro processo è ancora in stato di *working*: il processo resta nello stato *ready* e si blocca.
- Tutti i processi sono nello stato di *ready* e desiderano terminare correttamente: il processo entra nello stato di *committing*, trasferisce gli oggetti dalla copia delle intenzioni a quella originale e distrugge la copia delle intenzioni.
  Risveglia un altro processo che si era precedentemente bloccato; il processo svegliato completa la sua esecuzione.

Modello a scambio di messaggi

Processo coordinatore gestisce il descrittore dell'azione atomica. E' uno dei processi che realizzano l'azione atomica. Crea il descrittore dell'azione atomica ed attiva in parallelo tutti i processi partecipanti.

Two phase commit protocol

1. Il processo coordinatore, qualora sia disponibile a terminare con successo l'azione atomica, crea la copia delle intenzioni in memoria stabile ed esegue la *prepare* per entrare nello stato *ready*.
Invia ad ogni partecipante un messaggio di richiesta esito e rimane in attesa delle risposte.

2. Se arrivano una o più risposte di *esito negativo* l'azione deve essere abortita (passo 5). Se tutti i partecipanti rispondono con *esito positivo* l'azione deve terminare con successo (passo 3).

3. Viene eseguita la primitiva *commit* e viene inviato a ciascun partecipante un messaggio con l'indicazione di *terminare con successo*. Il coordinatore rimane in attesa del messaggio di *avvenuto completamento* da parte di tutti i partecipanti.

4. All'arrivo di tutti i messaggi di *avvenuto completamento* il coordinatore termina eliminando il descrittore dell'azione atomica.

5. Viene eseguita la primitiva *abort* e viene inviato il messaggio di *completare con aborto* a tutti i partecipanti. Il coordinatore termina abortendo a sua volta.

Ogni partecipante esegue il seguente algoritmo:

1. Terminate le operazioni sugli oggetti, il processo può aver abortito (*stato aborting*) o aver creato la copia delle intenzioni (*stato ready*). In entrambi i casi attende il messaggio *richiesta-esito* da parte del coordinatore.
2. Risponde con *esito positivo* o *esito negativo* a seconda dello stato in cui si trova. Se è in stato *aborting* termina abortendo, altrimenti resta in attesa del messaggio di completamento da parte del coordinatore.
3. Se viene ricevuto il messaggio *completare con aborto*, viene eliminata dalla memoria stabile la copia delle intenzioni ed il processo termina abortendo.
4. Se viene ricevuto il *messaggio completare* con successo, il processo termina correttamente trasferendo i valori della copia delle intenzioni nella copia originale degli oggetti ed elimina la copia delle intenzioni. Invia al coordinatore il messaggio *avvenuto completamento.*

•Nei sistemi distribuiti si possono avere due tipi particolari di malfunzionamenti:
- Perdita (o invalidazione) dei messaggi in rete
- Caduta dei singoli nodi della rete.

•Comportano il non arrivo a destinazione di alcuni messaggi o la non ricezione delle risposte da parte del processo mittente.

•Utilizzo di un **temporizzatore** per limitare il tempo di attesa di un messaggio. Al termine del tempo previsto, il processo viene riattivato e viene eseguita una procedura per richiedere nuovamente l'informazione oppure per abortire.

•Per fronteggiare la caduta dei singoli nodi, ogni processo partecipante deve mantenere in memoria stabile delle **variabili di stato** da utilizzare in fase di riattivazione.

AZIONI ATOMICHE NIDIFICATE

•In base alle proprietà di serializzabilità e tutto o niente, un'azione atomica rappresenta un'unità di programma che non può essere nidificata entro altre azioni atomiche.

•A1 e A2 azioni atomiche con A2 nidificata entro A1 (costituisce una delle operazioni eseguite da A1).

•Lo schema presenta problemi che non consentano, se non si modificano le proprietà delle azioni atomiche, la sua realizzazione.

Problemi

- Quando A2 termina, il processo che la esegue rilascia tutti gli oggetti su cui A2 ha operato. Gli oggetti sono disponibili per essere acquisiti da altri processi e quindi per operarvi all'interno di altre azioni atomiche.

➔ Violazione della proprietà di **serializzabilità** per quanto concerne l'azione atomica A1.

- Se A2 termina correttamente, essa modifica la copia stabile degli oggetti su cui ha operato e li rilascia.

Se A1 per qualche motivo non termina correttamente, abortisce e disfa tutte le modifiche effettuate sugli oggetti (**tutto o niente**).

Devono essere quindi ripristinati al valore iniziale anche gli oggetti di A2. Operazione difficile perché per la corretta terminazione di A2 questi valori sono andati perduti.

Possibilità di un **effetto domino** in quanto gli oggetti di A2 possono essere stati acquisiti da altre azioni atomiche.

Limitazioni

• Impossibilità di modularizzare un programma strutturato in termini di azioni atomiche.

• Alcune delle azioni atomiche nidificate all'interno di un'altra azione atomica potrebbero essere eseguite in concorrenza in quanto per effetto del **two phase lock protocol**, eventuali competizioni per l'uso di oggetti comuni sarebbero automaticamente risolte.

• Un'azione atomica nidificata potrebbe fallire senza implicare la terminazione anomala dell'intera azione atomica. Il programma potrebbe essere strutturato in modo da eseguire un'azione atomica alternativa (sottoazione)

Soluzione

• Modifica dei protocolli **two phase lock protocol e two phase commit protocol:**

1)Una sottoazione che termina con successo rende disponibili gli oggetti all'azione atomica (**solo a questa**) al cui interno la sottoazione è nidificata.

Si evita la **visibilità degli stati intermedi** di un'azione atomica a causa della corretta terminazione di una sua sottoazione.

2)Quando una sottoazione **termina con successo**, la seconda fase di commitment **viene ritardata** ed eseguita solo se l'azione più esterna **termina con successo**.

In caso di aborto dell'azione esterna, tutte le sottoazioni devono essere abortite. La modifica degli oggetti in memoria stabile è effettuata dall'azione atomica più esterna.

• Per garantire questo tipo di comportamento per ogni oggetto viene mantenuta aggiornata una **pila di copie di lavoro**.
All'inizio di una sottoazione viene generata una **nuova copia** in testa alla pila.

Se la sottoazione termina con **successo**, la copia in testa alla pila diventa la nuova copia di lavoro per l'azione più esterna; diversamente viene scartata la copia in **testa alla pila.**

CHIAMATA DI PROCEDURA REMOTA IN SISTEMI DISTRIBUITI

- Le azioni atomiche sono particolarmente utili per strutturare **sistemi transazionali** (accesso da parte di più utenti alle informazioni contenute in un data base)
- Meccanismo di **RPC** è **idoneo** per le comunicazioni tra un processo transazionale e i processi che geatiscono la base di dati (modello *cliente-servitore*). Il cliente, dopo la richiesta di servizio, **rimane in attesa della risposta** (*extended rendez-vous*)
- Il meccanismo RPC rappresenta il meccanismo più idoneo per la strutturazione di **sistemi transazionali distribuiti** (*client-server*)

- La realizzazione del meccanismo RPC in un **sistema distribuito** crea una serie di problemi legati ai malfunzionamenti propri di questi sistemi:
  - guasti alla sottorete di comunicazione
  - caduta dei singoli nodi
- Possibili malfunzionamenti:
  - messaggio di richiesta perduto
  - messaggio di risposta perduto
  - crash del nodo servitore durante il servizio
- Per evitare attese indefinite da parte del cliente, si ricorre ad un **meccanismo di temporizzazione** nella RPC (*time-out*)
- La **semantica** delle RPC dipende dalle azioni intraprese **allo scadere dell'intervallo di tempo**

- Le **proprietà delle azioni atomiche** consentono di ottenere **una** di queste possibilità **semantiche**:

  "at most once"

  ß
- In presenza di malfunzionamento durante la RPC questa **viene abortita senza produrre effetti**
- Per ottenere ciò, la RPC viene gestita come **un'azione atomica,** nidificata all'interno di un'azione di più alto livello, costituente l'operazione eseguita dal processo cliente
- E' possibile abortire la RPC, garantendo la sopravvivenza dell'intera azione svolta dal processo cliente, che può decidere di **scegliere strade alternative**

- RPC, vista come azione atomica nidificata, rappresenta **un'azione multiprocesso:**
  - processo che esegue la chiamata e si sospende
  - processo servitore creato per eseguire l'azione
- I due processi devono eseguire il **two-phome-committ protocol**, al termine **dell'intera operazione del processo cliente**.
- Esistono altre semantiche per la chiamata di procedura remota che non prevedono l'atomicità della chiamata

- Se il processo è interrotto dal meccanismo di temporizzazione prima della ricezione dei risultati, si invia di nuovo il messaggio di richiesta

⇓

- Possibilità di esecuzioni multiple della stessa procedura da parte del processo servitore

⇓

- Occorre che le procedure siano idempotenti (una loro ripetuta esecuzione produce sempre gli stessi risultati)

⇓

- In caso contrario è necessario ricorrere a particolari algoritmi per riconoscere e scartare i messaggi ripetuti