

# Azioni atomiche

## Definizioni

- **Azione atomica:** strumento di alto livello per strutturare programmi concorrenti e/o distribuiti, tolleranti a vari tipi di malfunzionamenti;
  - Realizzata con strumenti linguistici di linguaggi concorrenti
- **Consistenza dei dati:** ogni dato ha una relazione invariante, che lo caratterizza dal punto di vista semantico; *ogni operazione deve essere programmata in modo da lasciare l'oggetto su cui opera in uno stato consistente, cioè con l'invariante soddisfatta.*
  - Durante l'esecuzione, l'oggetto può passare attraverso stati inconsistenti, *ma questi non devono essere visibili dall'esterno.*
  - Monitor e Processi servitori
  - Una relazione di consistenza può anche riferirsi ad un *insieme di oggetti nel complesso*: la mutua esclusione garantisce l'atomicità dell'esecuzione sui singoli oggetti, ma non su tutto il sistema.
- **Azione atomica:** operazione che porta un insieme di oggetti da uno stato consistente S1 ad uno stato consistente S2.
- **Aborto:** distruzione degli effetti di una azione atomica e il ripristino dello stato iniziale.

## Proprietà

- **Tutto o niente (o atomicità nei confronti di malfunzionamenti):** Se  $S'$  è lo stato risultante dell'azione atomica, si deve avere  $S' = S1$  or  $S' = S2$ .
  - Tutto: terminazione normale, nuovo stato consistente
  - Niente: malfunzionamento, rollback
- **Serializzabilità (o atomicità nei confronti della concorrenza):** simile all'indivisibilità delle primitive, assicura che ogni azione atomica operi sempre su un oggetto il cui stato iniziale è consistente ed i cui stati parziali, durante l'esecuzione, non sono visibili ad altre azioni concorrenti. Condizioni (protocollo)
  1. Ogni oggetto deve essere acquisito da A in modo esclusivo prima di qualunque operazione su di esso;
  2. Nessun oggetto deve essere rilasciato prima che siano eseguite tutte le operazioni su di esso;
  3. Nessun oggetto può essere richiesto dopo che è stato effettuato un rilascio di un altro oggetto;
- **Two Phases Lock Protocol**
  - Fase crescente: si acquisiscono le risorse e si opera su di esse;
  - Fase calante: inizia appena è effettuato il primo rilascio e da qui non si possono più acquisire oggetti.
- **Condizione per il tutto o niente:**
  4. Nessun oggetto può essere rilasciato prima che l'azione atomica abbia completato la sua esecuzione (*i rilasci, quindi, sono le ultime operazioni dell'azione atomica*)
    - **Effetto domino:** l'aborto di una AA genera, come *effetto collaterale*, l'aborto di una diversa AA (e così via). E' causato da un rilascio di un oggetto prima di aver terminato l'elaborazione sugli oggetti e di aver quindi raggiunto uno stadio tale da garantire che, da quel punto in poi, qualunque evento anomalo accada, che l'azione non sarà più abortita.

## Commit

- Eseguita quando tutti gli oggetti sono al valore finale e produce come effetto l'impossibilità di aborto dell'azione atomica (**completamento con successo**).
- Requisito 4 riformulato: il rilascio si fa dopo il commit.
- **Transazione:** sequenza di operazioni effettuata su database che fanno passare il sistema tra due stati consistenti
  - **Atomicità**
  - **Consistenza:** passaggio da a stati corretti
  - **Isolamento:** serializzabilità
  - **Durata:** al completamento, effetti duraturi

## Terminazione anomala

- Cause
  - **Errore nei dati, eccezione:** esecuzione di primitiva abort
  - **Errore hardware o condizione di blocco critico:** in questo caso le informazioni residenti in RAM sono perse
    - **Ipotesi:** le informazioni su memoria di massa rimangono inalterate e le informazioni per il recupero vengono mantenute in memoria di massa.
- **Memoria stabile:** esiste la possibilità che le informazioni in memoria di massa vengano alterate, la memoria stabile è una astrazione con la proprietà di contenere informazioni necessarie al recupero e di non essere soggetta ad alcun tipo di malfunzionamento.
  - Realizzata mediante ridondanza
  - Le operazioni per leggere e scrivere su questa memoria sono atomiche (*stable read* e *stable write*)

## Stati di un processo

- **Working:** durante l'esecuzione del corpo dell'azione atomica
  - Oggetti in stato inconsistente
  - In caso di errore, bisogna abortire l'azione
- **Committing:** durante la terminazione corretta dell'azione e gli oggetti sono in stato finale. Si passa a questo stato con la primitiva *commit*
  - In caso di errore, il meccanismo di recupero deve completare l'azione
- **Aborting:** durante la terminazione anomala dell'azione; gli oggetti devono essere ripristinati al valore iniziale. Si passa a questo stato tramite primitiva *abort* o errore durante l'esecuzione
  - In caso di errore, si deve completare il ripristino dei valori iniziali

## Mantenimento dello stato

- Le informazioni sullo stato di un processo che sta eseguendo una azione atomica sono salvate in un **descrittore di azione** allocato in *memoria stabile*. L'azione inizia con la primitiva *begin action*.
- Gli oggetti sono salvati in *memoria stabile* e copiati in RAM quando ci si deve lavorare:
  - Aborto = distruzione della copia in RAM;
  - Terminazione = copia della RAM in memoria stabile.

- Terminazione: si crea prima in memoria stabile una **copia delle intenzioni**, cioè i valori finali degli oggetti in spazi di memoria diversi da quelli dell'oggetto stesso, così se cade a metà la terminazione posso avere la copia di intenzioni oppure abortire.
  - Se cado prima del commit, annullo l'azione (non ho copie di intenzioni consistenti);
  - Se cado dopo il commit, ho le copie di intenzioni e posso ricominciare a copiarle sugli oggetti.

## Schema di una azione atomica

```
// *****
// SCHEMA DI UNA AZIONE ATOMICA
// *****
// Begin action
<< creazione del descrittore di azione atomica (memoria stabile) >>;

// Fase crescente del Two Phases Lock Protocol

    << richiesta esclusiva degli oggetti residenti in memoria stabile >>;

    << creazione copie volatili degli oggetti >>;

// Corpo azione atomica

    << sequenza di operazioni sulle copie volatili >>;

    if (<< nessun malfunzionamento >>)
        terminazione(descrittore);
    else
        abort(descrittore);

    << distruzione copie volatili >>;

// Fase decrescente del Two Phases Lock Protocol

    << rilascio oggetti in memoria stabile >>;

// End action
<< eliminazione descrittore >>

// *****
// SCHEMA DI TERMINAZIONE
// *****
void terminazione(descrittoreAzione x)
{
    << creazione copia di intenzioni in memoria stabile >>;

    // Passaggio da working -> committing
    commit(x);

    << trasferimento valori da copia di intenzioni
        a oggetto in memoria stabile >>;

    << distruzione copia intenzioni >>;
}
```

## Realizzazione della memoria stabile

### Proprietà della memoria stabile

- Non è soggetta a malfunzionamenti
  - Utilizzo di ridondanza
- Le informazioni non vengono perse o alterate a causa della caduta dell'elaboratore
  - Operazioni di lettura e scrittura atomiche (*stable read/write*)

### Tipi di malfunzionamento

- **Errori in lettura:** rileggo e uso dei codici di ridondanza ciclica;
- **Errori in scrittura:** rilevo rileggendo ciò che ho scritto, si elimina l'errore riscrivendo;
- **Alterazione delle informazioni:** a causa di disturbi o guasti HW che perdurano per un certo numero di letture consecutive, utilizzo le *copie multiple* (ridondanza di livello due, cioè scritture su due dischi differenti)

### Realizzazione memoria stabile

- **Disco permanente:** astrazione ottenuta eliminando i guasti temporanei (errori in lettura e scrittura)
- **Memoria stabile:** elimina guasti hw e errori prodotti dalla caduta dell'elaboratore, si ottiene *con una coppia di dischi permanenti*.
- **Blocco stabile:** è una coppia di blocchi, uno per disco permanente
  - **Ipotesi di base:** *le due copie non vengono alterate entrambe dallo stesso malfunzionamento, quindi una delle due contiene sicuramente il valore corretto (valore del blocco)*

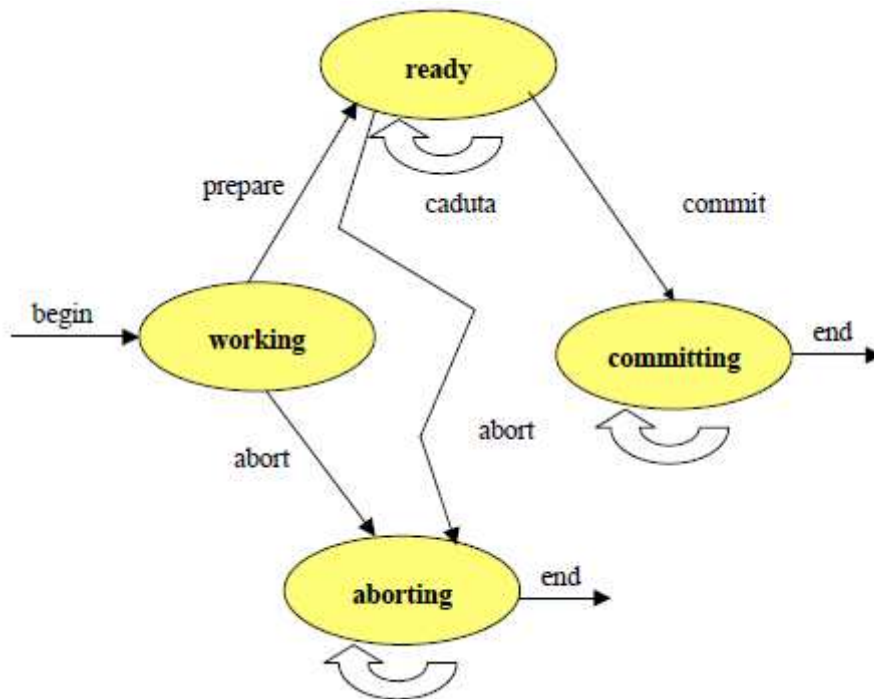
### Atomicità delle operazioni stable

- **Serializzabilità** ottenuta realizzando la memoria stabile come un monitor, che impedisce la divisibilità delle operazioni;
- **Tutto o niente**
  - Stable read: lo garantisce perché non effettua modifiche
  - Stable write: a seconda di quando avviene l'errore
    - Sulla scrittura del primo blocco: il secondo è ancora al valore originale, quindi annulla l'operazione;
    - Sulla scrittura del secondo blocco: il primo ha già il valore finale, quindi completa l'operazione;
    - In mezzo alle due operazioni: devo rileggere i due blocchi
      - Se uno è alterato, viene sostituito con l'altro
      - Se entrambi sono corretti, ma con valori diversi, c'è stato un guasto tra le scritture, quindi basta copiare il primo blocco sul secondo

### Azioni atomiche multi processo

- **Descrizione:** azioni atomiche che operano con più processi
  - **Ipotesi:** *i singoli processi operano ciascuno su oggetti diversi*
- **Proprietà:** perché sia atomica, devono esserci:
  - **Serializzabilità:** se ogni processo esegue il Two Phases Lock Protocol
  - **Tutto o niente:** tutti i processi devono terminare correttamente o abortire, se è un po' e un po' c'è inconsistenza

- **Stato ready:** processo che ha completato la propria sequenza di azioni ed è pronto a terminare con successo (*commit*), ma deve attendere che gli altri processi terminino le loro operazioni.
  - Si passa da *working* a *ready* con la primitiva *prepare*. Qui non posso più abortire unilateralmente, ma posso terminare con un *commit* o con un *abort*.
  - Se cado durante il *ready*, ripristino e torno in *ready*



### Two Phases Commit Protocol

1. Ciascun processo specifica la propria opzione (successo o aborto)
2. Prepare
3. Viene verificata l'azione di tutti: se tutti sono per completare con successo, si va a committing, altrimenti a aborting.

### Realizzazione nel modello a memoria comune: il descrittore di azione è un monitor

Processo iniziale:

1. Crea il descrittore dell'azione atomica
2. Crea i figli che eseguono l'azione atomica
3. Quando tutti han finito, riprende il controllo e cancella il descrittore

Processo i-esimo:

1. Richiesta esclusiva oggetti
2. Creazione copie volatili
3. Operazioni
4. Terminazione
  - a. Con aborto:
    - i. Entra in aborting
    - ii. Opzione riportata nel descrittore di periferica
    - iii. Risveglio dei processi ready;
  - b. Con successo:
    - i. Creazione copia delle intenzioni

- ii. Transito a ready attraverso prepare
- iii. Verifica se almeno uno ha abortito
  - 1. Se sì, distrugge copia intenzioni ed esegue abort
  - 2. Se no
    - a. Se ci sono ancora dei working, si sospende
    - b. Se tutti sono ready, entra in committing, copia dalle intenzioni all'oggetto e distrugge le intenzioni, poi risveglia il prossimo ready

### **Realizzazione nel modello a scambio di messaggi: il descrittore di azione è un oggetto del processo coordinatore**

Coordinatore:

1. Crea il descrittore dell'azione
2. Attiva i processi partecipanti
3. Se vuole terminare con successo
  - o Crea la copia di intenzioni ed esegue la prepare
  - o Invia un messaggio di richiesta esito a tutti i partecipanti
  - o Se arriva almeno una risposta negativa
    - Esegue abort
    - Invia a tutti i partecipanti il messaggio di completare con abort
  - o Se tutti danno ok
    - Si esegue commit
    - Si invia a ciascun partecipante un messaggio di completare con successo
    - Attende il messaggio di avvenuto completamento da tutti
    - Elimina il descrittore di azione

Partecipante:

1. Al termine delle operazioni, è in ready (e ha creato copia intenzioni) o aborting, ma comunque attende il messaggio di richiesta esito
2. Manda il suo messaggio:
  - a. Se è in aborting, dopo termina abortendo
  - b. Se è ready, attende il messaggio di completamento
    - i. Se riceve messaggio di completare con aborto
      1. Elimina la copia di intenzioni
      2. Termina abortendo
    - ii. Se riceve messaggio di completare con successo
      1. Copia da intenzioni a originali
      2. Elimina intenzioni
      3. Invia l'avvenuto completamento e termina

### **Azioni atomiche multi processo in rete**

- **Perdita di messaggi in rete:** utilizzo un temporizzatore, poi richiedo i dati o richiedo l'aborto
- **Caduta dei singoli nodi:** ogni partecipante deve avere in memoria stabile le informazioni di stato per ripartire correttamente alla riattivazione.

## Azioni atomiche nidificate

### Problemi

- Descrizione: A1 e A2 atomiche, con A2 nidificata in A1.
- Problema di serializzabilità: A2 termina correttamente, A1 poi non può farlo, ma non sa come annullare gli effetti di A2, perché A2 ha fatto tutto e quindi eliminato copie temporanee! Inoltre gli oggetti di A2 potrebbero essere già stati acquisiti da altri e modificati nuovamente...

### Soluzione: modifica dei protocolli 2PLP e 2PCP

- Se una sottoazione termina con successo, rende disponibili gli oggetti all'azione atomica padre (*solo a lei*)
- Se una sottoazione termina con successo, la seconda fase di committing è ritardata ed eseguita solo se l'azione più esterna termina con successo. In caso di aborto dell'esterna, tutte le sottoazioni abortiscono. Le modifiche in memoria stabile sono fatte solo dall'azione esterna.
  - Per garantire questo, per ogni oggetto viene mantenuta una **pila di copie di lavoro**
    - All'inizio di una sottoazione => nuova copia in testa alla pila
    - Al termine con successo => La copia in testa è la nuova copia di lavoro per l'azione esterna
    - Al termine con aborto => Testa scartata

## RPC in sistemi distribuiti

- RPC gestita come azione atomica per reggere i guasti (es. temporizzatore, a patto che siano azioni *idempotenti*)
- Semantica **at most once**: RPC abortita senza produrre effetti in caso di guasti
  - Azione atomica nidificata, con azione multi processo (chi esegue la chiamata e si sospende, servitore creato per eseguire l'azione) con esecuzione del 2PCP al termine dell'operazione del cliente