Deadlock e livelock - protocollo di locking a due fasi stretto

---

Index

- Introduction
- Soluzioni per il deadlock
  - Approcci risolutivi
  - Approcci preventivi
    - Esempio
- Livelock
- Soluzioni per il livelock
- Abort di una transazione
- Punto di commit
  - Dati sporchi
- Rollback a cascata
- Soluzione a dirty data
- Protocollo a due fasi stretto
  - Esempio
- Classificazione dei protocolli
  - Protocolli conservativi
  - Protocolli aggressivi
  - Protocolli a confronto

Introduction

Un **deadlock** si verifica quando ogni transazione in un insieme T è in attesa di ottenere un lock su un item sul quale qualche altra transazione nell'insieme T mantiene un lock, e quindi rimane bloccata, e quindi non rilascia i lock, e quindi può bloccare anche transazioni che non sono in T

Soluzioni per il deadlock

Esistono diverse soluzioni per il deadlock in base a che tipo di approccio utilizzano

Approcci risolutivi

Quando si verifica una situazione di stallo questa viene risolta

Per **verificare** il sussistere di una situazione di stallo si mantiene il **grafo di attesa**:

- nodi → le transazioni
- archi → $T_i \longrightarrow T_j$ se la transazione T_i è in attesa di ottenere un lock su un item sul quale T_j mantiene un lock
  Se in tale grafo c'è un ciclo si sta verificando una situazione di stallo che coinvolge le transazioni nel ciclo

Esempio >

Transazione	Risorsa	Stato
T_1	R_1	Bloccata
T_2	R_2	Bloccata
T_1	R_2	In attesa
T_2	R_1	In attesa

Grafo di attesa (Wait-For Graph):

- Nodo T_1: Ha un arco verso T_2 (attende R_2).
- Nodo T_2: Ha un arco verso T_1 (attende R_1).

Il grafo contiene un **ciclo**: $T_1 \rightarrow T_2 \rightarrow T_1$.

Per **risolvere** il sussistere di una situazione di stallo viene fatto un *rollback* su una transazione nel ciclo e successivamente viene fatta ripartire
Quando si parla di rollback nello specifico avviene:

1. la transazione è abortita
2. i suoi effetti sulla base di dati vengono annullati ripristinando i valori dei dati precedenti l'inizio della sua esecuzione
3. tutti i lock mantenuti dalla transazione vengono rilasciati

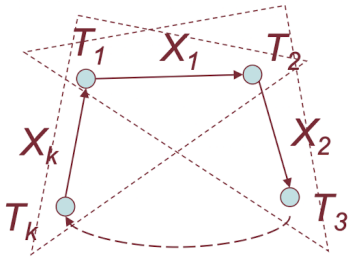
Approcci preventivi

Si cerca di evitare il verificarsi di situazioni di stallo adottando opportuni protocolli

Esempio

Si ordinano gli item e si impone alle transazioni si richiedere i lock necessari seguendo tale ordine. In tal modo non ci possono essere cicli nel grafo di attesa (e quindi non si può verificare un deadlock)

Se per assurdo supponiamo che le transazioni richiedono gli item seguendo l'ordine fissato e nel grafo di attesa c'è un ciclo



$X_1 \prec X_2$ (X_1 precede X_2 nell'ordinamento)
$X_2 \prec X_3$
...
$X_{k-1} \prec X_k$
da cui segue $\boldsymbol{X_1 \prec X_k}$

$\boldsymbol{X_k \prec X_1}$
(contraddizione)

Livelock

Si verifica un **livelock** quando una transazione aspetta indefinitivamente che gli venga garantito un lock su un certo item

una sorta di starvation ????

Soluzioni per il livelock

Il problema dell'attesa indefinita, può essere risolto in due modi:

- con una strategia *first came-first served*
- eseguendo le transazioni in base alle loro priorità e aumentando la priorità di una transazione all'aumentare del tempo in cui rimane in attesa

---

Abort di una transazione

Un abort può avvenire per una delle seguenti cause:

- La transazione esegue un'operazione non corretta (es. divisione per 0, accesso non consentito)
- Lo scheduler rileva un deadlock
- Lo scheduler fa abortire la transazione per garantire la serializzabilità (timestamp)
- Si verifica un malfunzionamento hardware o software

Punto di commit

Il **punto di commit** di una transazione è il punto in cui la transazione:

- ha ottenuto tutti i lock che gli sono necessari
- ha effettuato tutti i calcoli nell'area di lavoro

Vengono quindi esaurite tutte le situazioni che possono portare ad un deadlock, ma comunque i dati prima del commit sono sporchi (possono portare ad un rollback)

Dati sporchi

Per dati sporchi si intendono i dati scritti da una transazione sulla base di dati prima che abbia raggiunto il punto di commit

Rivediamo gli esempi >

Lock risolve → lost update (no aggregato non corretto, no dirty data)
Lock a due fasi risolve → lost update, aggregato non corretto (no dirty data)

Vediamo gli esempi
Ghost update con lock

T_1	T_2
wlock(X)	
read(X)	
X:=X-N	
write(X)	
unlock(X)	
	wlock(X)
	read(X)
	X:=X+M
	commit
	write(X)
	unlock(X)
wlock(Y)	
read(Y)	
Y:=Y+N	
commit	
write(Y)	
unlock(Y)	

Aggregato non corretto con locking a due fasi

T_1	T_3
	somma:=0
wlock(X)	
read(X)	
X:=X-N	
write(X)	
wlock(Y)	
unlock(X)	
	rlock(X)
	read(X)
	somma:=somma+X
read(Y)	
Y:=Y+N	
commit	
write(Y)	
unlock(Y)	
	rlock(Y)
	read(Y)
	somma:=somma+Y
	commit
	unlock(X)
	unlock(Y)

Rollback a cascata

Quando una transazione T viene abortita devono essere annullati gli effetti sulla base di dati prodotti:

- da T
- da qualsiasi transazione che abbia letto dati sporchi

Soluzione a dirty data

Per risolvere il problema della lettura di dati sporchi occorre che le transazioni obbediscano a regole più restrittive del protocollo di locking a due fasi

---

Protocollo a due fasi stretto

Una transazione soddisfa il protocollo di **locking a due fasi stretto** se:

1. non scrive sulla base di dati fino a quando non ha raggiunto il suo punto di commit (se una transazione è abortita non ha modificato nessun item sulla base di dati)
2. non rilascia un lock finché non ha finito di scrivere sulla base di dati (se una transazione legge un item scritto da un'altra transazione quest'ultima non può essere abortita)

Ciò mi permette di evitare di dover fare rollback sui dati

Esempio

Aggregato non corretto

T_1	T_3
	somma:=0
wlock(X)	
read(X)	
X:=X-N	
wlock(Y)	
read(Y)	
Y:=Y+N	
commit	
write(X)	
write(Y)	
unlock(X)	
unlock(Y)	
	rlock(X)
	read(X)
	somma:=somma+X
	rlock(Y)
	read(Y)
	somma:=somma+Y
	commit
	unlock(X)
	unlock(Y)

Dirty data

T_1	T_2
wlock(X)	
read(X)	
X:=X-N	
wlock(Y)	
read(Y)	
Y:=Y+N	
commit	
write(X)	
write(Y)	
unlock(X)	
unlock(Y)	
	wlock(X)
	read(X)
	X:=X+M
	commit
	write(X)
	unlock(X)

Classificazione dei protocolli

I protocolli si distinguono in:

- **conservativi** → cercano di evitare il verificarsi di situazioni di stallo
- **aggressivi** → cercano di processare più rapidamente possibile anche ciò che può portare a situazioni di stallo

Protocolli conservativi

Nella versione più conservativa una transazione T richiede tutti i lock che servono all'inizio e li ottiene se e solo se **tutti i lock sono disponibili**. Se non li può ottenere tutti viene messa in una cosa di attesa.
Si evita il deadlock, ma non il livelock

Per evitare il verificarsi sia del deadlock che del livelock occorre che una transazione T richiede tutti i lock che servono all'inizio e li ottiene se e solo se tutti i lock sono disponibili e nessuna transazione che precede T nella coda è in attesa di un lock richiesto da T

Vantaggi

- si evita il verificarsi del deadlock che del livelock
  Svantaggi
- l'esecuzione di una transazione può essere ritardata
- una transazione è costretta a richiedere un lock su ogni item che potrebbe anche se poi di fatto non lo utilizza

Protocolli aggressivi

Nella versione più aggressiva una transazione deve richiedere un lock su un item immediatamente prima di leggerlo o scriverlo
Può verificarsi un deadlock

Protocolli a confronto

Se la probabilità che due transazioni richiedano un lock sullo stesso item è:

- alta
  è conveniente un protocollo conservativo in quanto evita al sistema il sovraccarico dovuto alla gestione dei deadlock (rilevare e risolvere situazioni di stallo, eseguire parzialmente transazioni che poi vengono abortite, rilascio dei lock mantenuti da transazioni abortite)

- bassa
  è conveniente un protocollo aggressivo in quanto evita al sistema il sovraccarico dovuto alla gestione dei lock (decidere se garantire un lock su un dato item ad una data transazione, gestire la tavola dei lock, mettere le transazioni in una coda o prelevarle da essa)