Fault Tolerance: Salviamo lo Stato!

di Anna Pugliese

...che non è un «Dio salvi la Regina» made in Italy!!! In realtà l'argomento trattato questo mese si colloca nel cuore dei meccanismi per la tolleranza ai guasti dei sistemi d'elaborazione. Dopo un'esemplificativa presentazione del problema, illustreremo i principi di base delle tecniche più significative dello stato dell'arte.

Sul precedente numero di MC, avevamo parlato di Tolleranza ai guasti. Investigando sulla strutturazione del software, avevamo posto un po' di basi concettuali sulle quali fondare le strategie di «lotta al guasto». Ai fini della presente trattazione è utile richiamare quello che la volta scorsa avevamo chiamato, un po' impropriamente, ciclo di vita del guasto.

Un guasto, hardware o software, all'interno di un sistema, non può essere rilevato da nessun meccanismo, fintantoché esso non provoca errori; quando ciò accade, il sistema ha tempo per rilevare e ripristinare l'errore fino a che esso non causa un insuccesso.

La possibilità di ripristinare gli errori è, quasi sempre subordinata allo svolgimento di quell'attività che prende il nome di Salvataggio dello stato.

In generale

Stento a credere che qualcuno dei lettori sia sprovvisto della buona abitudine di richiamare, di tanto in tanto, dal menu del suo word processor, il comando di memorizzazione, per evitare di dover riscrivere daccapo il suo lavoro in seguito a cadute di tensione. Si intende: tanto di cappello a chi, terrorizzato dall'idea di sprecare il suo preziosissimo tempo, ha preferito acquistare il suo bravo gruppo di continuità. Ad ogni modo, sia gli uni che gli altri, sanno cosa sia una tecnica di salvataggio dello stato.

Se dovessimo applicare questo stesso principio ad un sistema in cui l'editing di un file non è l'unica applicazione in esecuzione, cioè ad un sistema in cui non è possibile stabilire, istante per istante, quali siano i dati che vengono aggiornati, non dovremmo fare nient'altro che memorizzare periodicamente il contenuto di tutti i file del sistema.

L'esecuzione periodica di backup (copie) totali del sistema, oltre ad essere una delle più antiche tecniche di salvataggio dello stato, è anche, a tutt'oggi, una delle più diffuse. La sua utilizzazione va dai semplici personal computer (si pensi al comando BACKUP delle ultime versioni dei sistemi operativi per personal computer) ai grossi centri di calcolo dove, sul far della sera, tecnici ed operatori in camice bianco montano e smontano pizze (mm! che odorino!, ndadp) su enormi unità a nastro che per intere ore registrano tutti i file del sistema.

Nonostante la sua diffusione, questa tecnica ha un pregio e due difetti. Il

```
(1)     X := read(conto_ i)
(2)     Y := read(conto_ j)
(3)     X := X+100000
(4)     write(X,conto_ i)
(5)     Y := Y-100000
(6)     write(Y,conto_ j)
```

Figura 1

```
(5.1)   PPC := 5                (5.1)   Y := Y-100000
(5.2)   Y := Y-100000           (5.2)   PPC := 5
```

(a) (b)

Figura 2

primo difetto è che essa, essendo una tecnica manuale, può essere applicata erroneamente; il secondo, più concettuale, è legato alla periodicità della sua applicazione: non tutto il lavoro è recuperabile, ma solo quello svolto prima dell'ultimo backup. Il suo unico pregio è tuttavia fondamentale: questa tecnica è più economica delle altre.

Esistono dei casi in cui è comunque necessario utilizzare tecniche migliori. In generale essi sono due: quando la parte irrecuperabile di lavoro vale più del risparmio ottenuto, e quando il lavoro perso non può essere rieseguito (si pensi alle applicazioni da cui dipendono vite umane: shuttle, centrali nucleari e affini...).

Nel passaggio dalla tecnica del backup periodico totale ad altre tecniche di salvataggio dello stato, il salto è netto: occorre prevedere meccanismi da applicare in fase di programmazione, del software di base e/o di quello applicativo, mediante i quali le operazioni eseguite sul sistema si preoccupano, oltre che di provocare transizioni di stato del sistema, anche di assicurare che lo stato sia memorizzato su supporti di memoria sicuri, dove il termine sicuro è quasi sempre sinonimo di ridondante.

Atomicità

Per procedere con maggiore chiarezza, è il caso di sceglierci un esempio sufficientemente significativo. Osserviamo allora la figura 1.

Essa mostra un pezzo di programma Pascal-like, composto da 6 istruzioni, che agisce su due conto correnti, per realizzare un versamento di L. 100.000 sul conto i prelevando la somma da versare, dal conto j.

Supponiamo che un errore si verifichi in seguito all'esecuzione dell'istruzione 5. Qui, il termine errore, è usato nella sua più corretta accezione di «erroneo assegnamento di un valore allo stato interno del sistema». Per esemplificare, l'errore può consistere nella perdita di tutta la parte dello stato residente in memoria principale a seguito di una caduta di tensione. Indichiamo questo tipo di errore con il termine CRASH del sistema.

In seguito al crash, è necessario eseguire un reboot cioè un ricaricamento del sistema. È evidente che i dati disponibili sono solo quelli che erano presenti su supporti di memoria di massa, precedentemente al crash, essendo diventati non significativi i dati presenti in memoria principale. Su tali supporti, precedentemente al crash, il valore del conto i era stato incrementato di 100.000, mentre nessuna variazione aveva subito il valore del conto j. Ora poiché non abbiamo nessun dato mediante il quale è possibile determinare se il crash è avvenuto prima o dopo l'esecuzione dell'istruzione 5, il risultato è che ci troveremo davanti ad un'incongruenza dei dati del sistema.

Una soluzione abbastanza ovvia, consiste nel mantenere traccia, durante l'esecuzione, di quali operazioni sono state eseguite e di quali devono ancora essere eseguite. Nel caso dei nostri conto correnti, basterebbe mantenere in memoria secondaria un'informazione riguardante l'ultima operazione che è stata eseguita. Al momento del crash questa informazione varrebbe «5», sicché, in seguito al reboot, una volta letto il valore 5 dalla memoria di massa, l'algoritmo di recovery provocherebbe l'esecuzione del programma a partire dall'istruzione 5+1.

È inutile osservare che l'istruzione 5+1 cioè l'istruzione 6, ha bisogno di conoscere il valore della variabile Y, essendo più che ovvia la necessità di aver memorizzato tutte le variabili in memoria di massa. Ma questo non è assolutamente un problema. Il problema vero è un altro, e mi piacerebbe proprio conoscere il lettore che, pur non avendo mai sentito parlare di atomicità, si è accorto dell'inghippo presente nella soluzione proposta.

OK! Visto che i 5 minuti di tempo a disposizione per scoprire l'inghippo sono scaduti, vediamo di ragionarci sopra insieme.

Proviamo ad implementare la soluzione incriminata. Sia PPC una variabile (Pseudo Program Counter), mantenuta costantemente aggiornata su un supporto di memoria di massa, ed il cui valore indica il numero dell'ultima istruzione eseguita. In figura 2 sono riportati i due possibili modi di sostituire l'operazione 5 con un pezzo di programma che

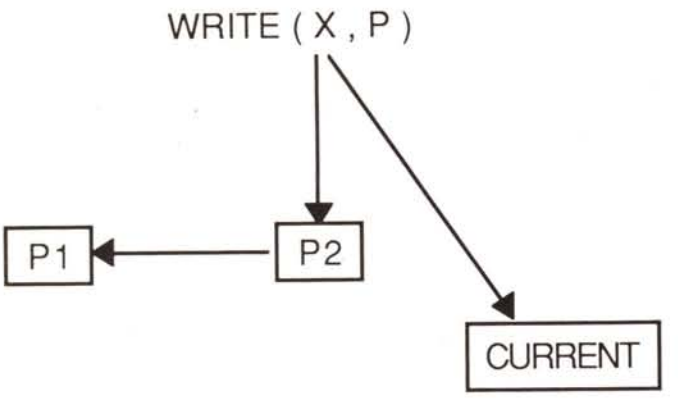

Figura 3

aggiorni anche PPC. In modo analogo saranno modificate le altre operazioni del pezzo di programma in figura 1.

Supponiamo che il crash del sistema sopraggiunga dopo l'esecuzione dell'operazione 5.1, ma prima dell'esecuzione di 5.2. Nel caso (a), dopo il reboot sarà eseguita l'operazione 6, provocando l'assegnamento a conto_j dello stesso valore posseduto precedentemente mentre contro_i era stato correttamente incrementato di 100.000. Nel caso (b), dopo il reboot, il valore di PPC sarà 4, non essendo ancora stato aggiornato dall'istruzione 5.2; l'esecuzione riprenderà allora dalla istruzione 4+1=5; in altre parole l'istruzione 5.1 sarà eseguita due volte sicché la somma prelevata da conto_j risulterà essere 200.000 invece di 100.000.

Pare che un ricercatore di una grossa azienda americana, abbia proposto a questo punto di utilizzare un secondo pseudo contatore di programma, col risultato di essere stato misteriosamente trasferito dal mese successivo a lavorare alla fotocopiatrice.

Una soluzione corretta può essere trovata solo avendo a disposizione la possibilità di eseguire alcune operazioni in maniera atomica.

L'atomicità è la possibilità di eseguire qualcosa in modo tale che in seguito all'esecuzione

— tutti i risultati dell'esecuzione sono correttamente ottenuti

oppure

— l'esecuzione non produce alcun effetto sullo stato del sistema.

L'aver utilizzato il vago termine «qualcosa» per indicare ciò che si vuole eseguire, è dovuto al vasto campo di applicazione dell'atomicità. In altri termini, non importa se ciò che si vuole eseguire è una sola operazione oppure un insieme di operazioni, quello che importa è che il risultato dell'esecuzione dev'essere tale da cadere in uno dei due casi sopra prospettati. Questa proprietà è detta del TUTTO o NIENTE. In realtà essa non basta per assicurare l'atomicità, ma ai fini della presente trattazione è conveniente trascurare la differenza esistente tra atomicità e proprietà del tutto o niente.

Ammesso e non concesso (visto che non abbiamo ancora visto come questa atomicità può essere effettivamente implementata) di avere la possibilità di eseguire qualcosa in maniera atomica, il nostro problema dei conto correnti è risolto in maniera banale: basterà eseguire atomicamente la sequenza composta dalle due operazioni 5.1 e 5.2 della figura 2, siano esse poste nell'ordine (a) o nell'ordine (b).

Analogamente saranno eseguite tutte le altre operazioni presenti in figura 1; ognuna di esse sarà tentata e ritentata fino a quando non sarà stata correttamente eseguita. E se i PPC ci sono antipatici e vogliamo una soluzione ancora più semplice, perché no, basterà eseguire in maniera atomica l'intera sequenza delle 6 operazioni. Insomma, c'è poco da dire: questa benedetta atomicità è proprio una bella invenzione. Non ci resta altro che implementarla.

Un esempio di memoria stabile

Un'operazione di scrittura in memoria secondaria, consiste generalmente nell'invio di un certo numero di byte al driver di I/O interessato, il quale a sua volta provvederà ad intraprendere una serie più o meno lunga di azioni per far sì che tali informazioni vengano fisicamente memorizzate. Niente di più facile che un qualsiasi imprevisto, provochi la sospensione di questa serie di azioni. I programmatori più accaniti, sanno dove andare a mettere le mani per scoprire, al termine di un'operazione di scrittura, se essa è andata a buon fine oppure no, ed i computer più cattivi sono ormai abbastanza buoni da non tenere nascosti simili tristi eventi.

Il problema vero è che capita a volte di non trovare più in memoria il vecchio valore, avendo subito quest'ultimo una serie di trasformazioni che pur non avendo prodotto un risultato attendibile, hanno fatto abbastanza per non rovinare quel poco di attendibile che c'era. In parole semplici, quello che è successo è né tutto né niente.

In simili casi, non si tratta di atomizzare una sequenza di operazioni, ma una singola operazione, il che è concettualmente identico, in quanto quest'ultima è realizzata a sua volta da una sequenza di operazioni di livello inferiore. Esistono per fortuna dei livelli del sistema tanto bassi da permettere l'esecuzione di operazioni in maniera non ulteriormente divisibile, cioè in maniera atomica, si pensi all'assegnamento di una costante ad un registro del linguaggio macchina. Qui, il tutto o niente è conseguenza diretta del fatto che l'operazione è fisicamente indivisibile. Il problema a questo punto, è quello di approfittare dell'indivisibilità di queste operazioni e scrivere algoritmi che permettono di simulare l'indivisibilità anche ai livelli superiori.

Torniamo alla nostra operazione di scrittura in memoria secondaria. Chiamiamo P la pagina sulla quale agisce tale operazione. Alla pagina logica P facciamo corrispondere le due pagine fisiche P1 e P2. Infine utilizziamo una variabile che chiameremo Current nella quale memorizzare uno dei due valori, «P1» o «P2», specificanti quale fra le due pagine contiene il valore da assumere come quello corrente della pagina P.

Per evitare di perderci dietro ad una lunga serie di procedure, descriviamo schematicamente l'algoritmo da utilizzare, facendo riferimento allo schema presente in figura 3.

L'operazione di scrittura del dato X sulla pagina P, può essere implementato mediante l'operazione Write (X,P), che agisce sulla pagina fisica P2 e sulla variabile Current.

Supponiamo che il valore di P sia contenuto fisicamente nella pagina P1. L'operazione di scrittura si incarica di

memorizzare sulla pagina P2 il nuovo valore X da assegnare a P. Se questa operazione non va a buon fine, opportune procedure di recovery provvederanno a far ripartire l'esecuzione dal punto più conveniente, tenendo presente che il valore di P1 è un valore significativo per P.

Se l'operazione di aggiornamento di P2 viene completata con successo, sarà ulteriore preoccupazione della WRITE, registrare tale evento ponendo il valore «P2» nella variabile Current. Fino a quando questo non non si verifica, il corretto valore per P continua ad essere quello di P1.

Non appena Current assume il valore «P2», si dice che l'operazione ha raggiunto il suo punto di Commit o di «non ritorno»; d'ora in poi ulteriori opportuni algoritmi si preoccuperanno di trasferire il valore di P2 nella pagina P1.

È doveroso richiamare l'attenzione sul modo in cui il valore Current è mantenuto sul sistema. I modi possibili sono tanti, e dipendono dal tipo di guasti e, conseguentemente, di errori cui si vuole far fronte.

Il modo migliore sarebbe quello di mantenerlo in posizioni di memoria che siano aggiornabili con operazioni indivisibili. Per far fronte alla volatilità di tali posizioni di memoria si potrebbero usare copie multiple del valore di Current, aggiornandole mediante l'invio di messaggi di tipo broadcast atomico. Ma anche se Current fosse mantenuto su memoria di massa, si pensi a paginate di valori di tipo Current, il suo aggiornamento non creerebbe gli stessi problemi che abbiamo cercato di risolvere, essendo comunque presenti sul sistema, all'atto dell'aggiornamento di Current, sia il vecchio che il nuovo valore di P (P1 e P2).

Al di là dell'esempio cui abbiamo accennato, cerchiamo di riassumere i punti forti della strategia utilizzata. Questi sono fondamentalmente due:

— avere a disposizione un'area di lavoro sulla quale effettuare scritture di dati aggiornati, senza perdere i valori precedenti dei dati.

— Utilizzare algoritmi di COMMIT, basati su un'azione indivisibile in seguito alla quale il valore da considerare come quello corrente è il nuovo valore presente nell'area di lavoro.

L'applicazione di questa strategia di base, porta alla definizione di tecniche di salvataggio dello stato.

Di seguito, una di tali tecniche, sufficientemente diffusa, è presentata in una versione alquanto schematica data la ristrettezza dello spazio a disposizione.

Il log delle modifiche

Consideriamo una sequenza di istruzioni, della quale si desidera una esecuzione atomica.

Indichiamo semplicemente con S, lo stato interno posseduto dal sistema prima dell'esecuzione della sequenza di istruzioni.

Poiché ogni istruzione della sequenza modifica il valore dello stato interno, utilizziamo per tenere traccia di tali modifiche una struttura dati che permetta di registrare, per ogni istruzione, il vecchio ed il nuovo valore dello stato.

L'insieme delle strutture dati corrispondenti all'insieme delle istruzioni della sequenza, sarà poi collegato a lista alla struttura contenente lo stato corrente del sistema. Il tutto è illustrato in figura 4.

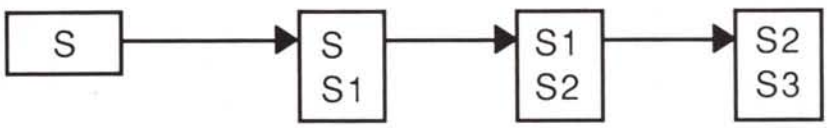

Figura 4

Il protocollo di Commit, si farà carico, al termine dell'esecuzione della sequenza di istruzioni, di stabilire il passaggio del valore dello stato, che per tutta la durata dell'esecuzione è S da S ad S3. Il vantaggio offerto da questa tecnica consiste nell'avere a disposizione la sequenza dei valori intermedi assunti dallo stato, il che, in seguito ad eventuali errori, potrebbe essere d'aiuto per evitare di dover ripartire daccapo nell'esecuzione.

La tecnica del log (la cui traduzione corrisponde più o meno a «giornale di bordo»), è spesso applicata con alcune varianti.

Una di esse, detta log degli UNDO, consiste nel registrare nelle strutture concatenate a lista, non già la coppia di valori vecchio/nuovo, ma delle istruzioni di UNDO, cioè di istruzioni capaci di disfare la transizione di stato provocata dall'istruzione eseguita. Se, ad esempio, l'istruzione da eseguire è X:=X+1, tale istruzione viene realmente eseguita sulla struttura dati contenente il valore di X, ma a tale struttura verrà collegata a lista un'altra struttura contenente l'istruzione X:=X−1, che è proprio l'istruzione di UNDO corrispondente. Le possibili varianti della tecnica dei log, così come, più in generale, tutte le possibili tecniche di salvataggio dello stato, debbono essere attentamente valutate in relazione alle caratteristiche possedute dal problema che si vuole risolvere. In astratto nessuna di esse è migliore delle altre.

Conclusioni

La strategia del log delle modifiche, è stata scelta in questa trattazione grazie alla sua diffusione fra programmi applicativi di alto rango. Mi riferisco a Sistemi per la Gestione di Basi di Dati, commercializzati per grossi Mainframe delle più note industrie di computer. Del resto, la necessità di ricercare strategie e meccanismi di salvataggio dello stato, così come, più in generale, meccanismi per la tolleranza ai guasti, si è manifestata storicamente proprio nello sviluppo di questo tipo di software. Nel seguito, tale ricerca è divenuta una degli scopi principali di un'altra area dell'informatica, che è quella dei sistemi distribuiti. Sempre per continuare questa passeggiata storica, quello che accadde nel seguito fu la scoperta che nonostante l'apparenza, le soluzioni cui pervenivano i ricercatori delle due diverse aree, erano basate su principi comuni, non solo fra di loro, ma addirittura con quei principi che reggevano le basi della produzione componentistica affidabile. Le conseguenze di questa scoperta hanno portato negli ultimissimi anni, alla progettazione di sistemi in cui le strategie di tolleranza ai guasti non sono applicate separatamente alla progettazione dei vari livelli del sistema, ma realizzate a parte per essere poi utilizzate uniformemente dai progettisti. Avere la possibilità di lavorare su simili sistemi, significa poter utilizzare potentissimi costrutti, messi a disposizione dal linguaggio offerto dal sistema, mediante i quali l'affidabilità è garantita in maniera trasparente. L'idea che sta alla base di questi sistemi è quella delle Transazioni Atomiche, intese come strumento per la risoluzione integrata del problema dell'affidabilità e di quello della concorrenza. Ma su questo torneremo in futuro. MC

SIM-HI·FI-IVES

22° salone internazionale della musica e high fidelity
international video and consumer electronics show

8-12 settembre 1988
Fiera Milano

STRUMENTI MUSICALI,
ALTA FEDELTÀ,
HOME VIDEO,
HI-FI CAR,
CAR ALARM SYSTEM,
PERSONAL COMPUTER,
VIDEOREGISTRAZIONE,
ELETTRONICA DI CONSUMO.

Ingressi per il pubblico:
Piazza Carlo Magno
Via Gattamelata
Reception operatori:
Via Gattamelata
(Porta Alimentazione)
Orario: 9.00 - 18.00
Aperto al pubblico:
8-9-10-11 settembre
Giornata professionale:
lunedì 12 settembre

HOME VIDEO
3ª Rassegna delle videocassette registrate

Segreteria Generale SIM-HI.FI-IVES:
Via Domenichino, 11 - 20149 Milano
Tel. 02/4815541 - Fax 02/4696055 - Telex 313627

VIVA i Giovani 88
Festa per i giovani musicisti