Soluzioni di *clustering*, replicazione e *stand by server*

Giuseppe Sacco

5 maggio 2006

Sommario

La libera traduzione italiana della parola *cluster* è *ammasso* oppure *agglomerato*. In informatica si usa da decenni per indicare una collezione di tracce di dischi. Più di recente è invece utilizzato per *agglomerato di computer* che può avere varie conformazioni, ma il fine è sempre lo stesso: aumentare la resilienza del sistema, cioè fare in modo che il servizio fornito da quelle macchine sia il più continuo possibile.

Ci sono vari tipi di *cluster*, come ad esempio quello relativo ai dischi. L'avere più dischi che vengono visti come un solo volume permette di avere *file system* più grandi del singolo disco, di sopportare la rottura di uno o più dischi senza perdere dati, di accedere a maggior velocità rispetto al singolo disco. Questi *cluster* implementano tecnologie chiamate RAID e sono spesso utilizzati in meccanismi di SAN *Storage Area Network* o NAS *Network attached storage*

Nel caso di computer, visti come unità di calcolo, questo *agglomerato* può avere vari scopi

- bilanciare il carico di lavoro tra più macchine perché una singola macchina non lo reggerebbe;
- ottenere una alta affidabilità del servizio, con una seconda macchina sempre pronta a prendere il posto di quella che offre un certo servizio;
- avere una macchina con dati aggiornati "in ritardo" allo scopo di consentire di recuperare i dati persi per "l'errore umano."

1 Bilanciamento del carico

Il bilanciamento del carico avviene in maniera pilotata dall'hardware, dal sistema operativo oppure dall'applicazione.

Se si utilizzano macchine perfettamente eguali, ad esempio dei server web con documenti statici, allora è possibile fare in modo che lo *switch* al quale queste macchine sono collegate, le chiami a turno, oppure lo stesso *switch* può conoscere il carico dei computer per instradare la richiesta sul server più idoneo.

Una seconda possibilità è usare un DNS che fa corrispondere allo stesso nome `www.mioserver.it` più di una macchina. I DNS hanno la capacità di offrire i nomi secondo la tecnica del *round robin*.

Si può far fare il bilanciamento al kernel linux utilizzando il progetto `LVS` (`http://www.linuxvirtualserver.org/`) che prevede una macchina *director* equivalente dello *switch* hardware e una serie di server applicativi. Questi server possono essere sia macchine Linux che Unix che Windows, a seconda del tipo di implementazione. Ci sono tre diversi modelli implentativi: LVS-NAT (*network address translation*) permette sistemi operativi eterogenei ma è il più lento, LVS-Tun (*tunnelling*) richiede linux ma permette di avere i server in diverse LAN/WAN, LVS-DR (*direct routing*) è il più veloce ma le macchine devono essere nella stessa LAN. Per questo tipo di *cluster* e per tutti quelli che hanno una macchina di controllo, come il *director*, si tenga presente che il controllo non è una operazione che richiede potenza di calcolo, quindi per questo ruolo è sufficiente un computer non particolarmente veloce.

Se l'applicazione è in grado di gestire autonomamente il bilanciamento del carico, allora è possibile fare in modo che vari server condividano le sessioni di lavoro. Un esempio è quello di `JBoss`, un application server per java, che permette di gestire *cluster* che gestiscano la sessione. (Se l'applicazione è **stateless**, cioè senza stato, allora la sessione non ha importanza e il *cluster* può tranquillamente essere

gestito dal sistema operativo. Se invece è **statefull**, cioè con stato, allora lo stato deve essere memorizzato con i meccanismi del *cluster* perché non sia relegato ad un solo nodo.) Altro esempio è quello di `WebSphere` che definisce tre tipi di *cluster* proprio in base alla gestione della sessione, anzi, alla *session affinity*. La versione 6 di `WebSphere` include quello che si chiama *eXtended Deployment*: un ingranaggio della nuova parola d'ordine, "grid," che permette di spostare intere applicazioni da un nodo all'altro in maniera trasparente per bilanciare il carico, ma anche in maniera preventiva utilizzando le statistiche di carico dell'ultimo periodo.

L'assegnamento di un un client ad un nodo del *cluster* può essere costante, nel senso che viene fatto al login e mai cambiato, oppure può variare nel tempo, spostando la sessione dell'utente da una macchina all'altra. Il primo caso va bene se tutte le sessioni di lavoro sono eguali (stessa durata, stesso "peso" computazionale), ma non va bene nel caso di sessioni lunghe e varie come quelle di lavoro su un ERP. In questo secondo caso spesso il bilanciamento è fatto staticamente sull'application server e dinamicamente sul database. In genere la gestione del bilanciamento delle connessioni ad un database avviene con i *pool* di connessioni o suoi derivati (come i MultiPool che `WebLogic Server` usa per `Oracle RAC`) che permettono di avere tante connessioni aperte in contemporanea con i vari nodi del *cluster*, selezionando una connessione diversa per ogni transazione. Un *pool* è un insieme si connessioni dall'applicazione al database; queste connessioni dopo l'uso non vengono chiuse, ma rimangono aperte e usate per richieste successive in modo da diminuire il tempo di connessione. Inoltre le connessioni possono essere aperte verso tutti i nodi del *cluster* e usate in maniera circolare, in modo da inoltrare le richieste a turno ai vari nodi.

1.1 Database

Una delle soluzioni open source è `pgpool` che si occupa del bilanciamento del carico per postgresql. `pgpool` è capace di collegarsi a due diversi database postgresql e di offrire un connection pool al client. Ogni transazione verrà effettuata su entrambe le macchine, mentre le *query* esterne alle transazioni verranno distribuite a turno tra le macchine.

Un'altra soluzione open source, più sofisticata, è `pgcluster` che permette di avere un ambiente *multi master* vale a dire che ogni transazione viene cominciata in uno qualsiasi dei nodi che poi si sincronizza con gli altri. `pgcluster` è costituito da almeno 4 macchine: una che bilancia il carico tra i vari nodi, almeno due nodi, una di gestione della replicazione. Il numero massimo di nodi è 128 e non è necessario avere uno storage condiviso.

Per chi fosse interessato a `MySQL`, esiste una implementazione nativa di *cluster* che richiede uno o più nodi *sql* per la gestione delle richieste, uno o più nodi *data* con i dati veri e propri (che devono risiedere interamente in RAM), un nodo per la gestione del cluster. Questa implementazione sfrutta i nodi *data* come sistema di memorizzazione e non utilizza affatto il disco. Per avere la replicazione si devono raddoppiare le macchine usate per i nodi *data*. Allo spegnimento del cluster ogni dato viene perso, quindi di norma si fa un backup da ricaricare all'accensione. Il bilanciamento del carico tra i vari nodi *sql* va fatto a livello applicativo, oppure tramite *connection pool*.

Una nota a parte va fatta per `Oracle RAC` che ha la caratteristica di richiedere uno spazio disco condiviso tra le varie istanze del *cluster*: per implementare un RAC su varie macchine è necessario che tutte accedano allo stesso disco in contemporanea. Questa soluzione è in genere parecchio costosa e non è stata seguita da nessun altro DBMS.

1.2 Calcolo parallelo

All'interno della sezione sul bilanciamento del carico va fatta una parentesi sui *cluster single system image*, cioè quei cluster che visti da fuori si comportano come se fossero una sola macchina, usati per il *High Performance Computing* (contrapposto a *High Availability*) Si tratta di unire la potenza di parecchie macchine per effettuare calcoli che di norma non richiedono molti accessi al disco, ma invece sfruttano il parallelismo e utilizzano lo scambio di messaggi per far comunicare tra loro i vari processi. Si tratti quindi quasi sempre di applicazioni scientifiche studiate appositamente per il funzionamento su macchine parallele. Le due soluzioni più famose di questo tipo sono `openmosix` e `beowulf`. Ciascuna di queste soluzioni include anche un *file system* distribuito, ma bisogna ammettere che normalmente le applicazioni parallele fanno veramente rari accessi al *file system* e quindi è sufficiente ave-

re una o due macchine che offrano il servizio di *file system*.

Sia `openmosix` che `beowulf` sono prodotti maturi e si occupano principalmente di bilanciare il carico tra le varie macchie, spostando i processi da un nodo ad un'altro. Hanno entrambi anche un meccanismo di riconfigurazione dinamica che permette l'aggiunta o la rimozione di un nodo in qualsiasi momento.

2 Alta affidabilità

L'alta affidabilità o *high availability* è un concetto che indica la disponibilità costante nel tempo di un certo servizio. Il meccanismo del *cluster* serve proprio a permettere questa continuità, facendo in modo che se la macchina principale si guastasse, la si possa sostituire con una seconda macchina durante il periodo di fermo. Il passaggio del servizio, detto *failover*, da una macchina all'altra può essere automatizzato o gestito manualmente, a seconda delle necessità. Difatti la configurazione di un *cluster* HA non è mai semplice e a volte il gioco non vale la candela.

Ci sono molte realtà nelle quali si fornisce un servizio che può anche mancare per pochi minuti senza generare troppi problemi. In tutti questi casi è possibile avere un piccolo *cluster* molto semplice nel quale esiste una macchina principale che offre il servizio e una secondaria che copia periodicamente i dati dalla prima. In caso di guasto della prima si può fare in modo che la seconda, in maniera automatica o manualmente, si faccia carico dei servizi.

Questa soluzione ha come unico "dettaglio implementativo" quello di scegliere come mantenere aggiornata la seconda macchina. Lo si può fare sia a livello di *file system*, con `rsync`, sia a livello di device, con `nbd`.

2.1 Sincronizzazione

`rsync` è uno strumento che copia un *file system* da una macchina ad un'altra. La prima volta la copia avviene nella maniera canonica, ma dalla successiva vengono copiati solo le sezioni modificate dei file cambiati. In questo modo il tempo di aggiornamento della seconda macchina è proporzionale solo alle modifiche fatte sulla macchina originale. In molte situazioni `rsync` è velocissimo e può essere eseguito anche ogni minuto, ma in altre non è possibile utilizzarlo poiché i dato non sono memorizzati su *file system*, come nel caso di volumi logici assegnati ai database, oppure nel caso di informazioni che riguardano gli *inode* che non verrebbero copiati da `rsync`.

`nbd` è un modulo del kernel di Linux. Si tratta di una sigla *Network Block Device* che indica appunto la natura del modulo: un device a blocchi via rete. Con nbd è possibile utilizzare il disco di una macchina remota come se fosse un disco locale. L'utilizzo che se ne fa in questi casi è di fare vedere alla macchina principale sia il disco locale che quello remoto, in modo da fare un *mirror* tra i due. Il vantaggio di `nbd` è che ogni singola scrittura avviene su entrambe le macchine in contemporanea, quindi la seconda macchina è sempre aggiornata. La sua configurazione è però più complessa di quella di `rsync`. Inoltre la connessione usata da `rsync` può essere compressa e cifrata, mentre quella di `nbd` non può esserlo.

2.2 Controllo

Il passaggio automatico di un servizio da un nodo all'altro è gestito su GNU/Linux da un programma chiamato `heartbeat` oppure da soluzioni proprietarie che ogni sistema operativo offre, come `HACMP` su AIX, o ancora da società che non vendono hardware, come Veritas. Tutte queste soluzioni prevedono la definizione di risorse che possono essere: programmi applicativi, indirizzi IP, dischi, *file system*, o altro ancora. Ogni risorsa è destinata ad un particolare nodo (che di norma ne gestisce più di una) e può essere fatta migrare su altri nodi in caso di necessità.

La gestione automatica dell'alta affidabilità richiede un meccanismo per la verifica dello stato dei nodi. In genere questo utilizza una connessione fisica privilegiata (un cavo di rete esclusivamente usato per HA) tramite il quale il software del *cluster* comunica tra i vari nodi e verifica lo stato delle risorse. Se una risorsa o un nodo non rispondono entro certi periodi allora un altro nodo si fa carico del problema.

La normale implementazione di una applicazione in *cluster* HA prevede di norma i seguenti passi

1 creazione di un IP virtuale tramite il quale fare raggiungere l'applicazione;

2 installazione dell'applicazione come risorsa del *cluster*;

3 installazione dei dati dell'applicazione come risorsa del *cluster*;

4 scrittura di un programma che controlli se l'applicazione funziona correttamente;

5 configurazione di HA per l'utilizzo del programma 4. e dei meccanismo di *failover*;

Il programma HA farà in modo da dare allo stesso nodo le tre risorse: IP virtuale, dati e applicazione. Inoltre continuerà a controllare l'applicazione in maniera periodica con il programma fornito. In caso di problemi le tre risorse verranno spostate secondo le politiche di *failover* su un altro nodo.

3 *Stand by database*

Spesso la necessità di un cluster è relativa ai dati memorizzati in un database. Alcuni di questi database hanno una caratteristica interessante legata al fatto che internamente funzionano con un meccanismo di *journaling*. Il giornale dei DBMS (database management server) è in genere un file, o più file, nel quale vengono scritte tutte le transazioni andate a buon fine. Scrivendo in un solo file i DBMS terminano questa scrittura relativamente in fretta. Periodicamente, con una operazione chiamata spesso *checkpoint*, il contenuto di questi file viene copiato sui veri e propri file dati.

Questi piccoli file (che postgresql chiama WAL *write-ahead log*, informix chiama *transaction log* e oracle chiama *redo log*) possono essere usati anche durante l'operazione di ripristino di un database. Al ripristino si prendono i file dati non aggiornati e una serie di *log* più recenti e si riapplicano tutte le modifiche ai file dati. Informix ha una tecnologia propria, chiamata HDR *high availablity data replication*, che fa esattamente questo: attende che il log sia completo e lo manda via rete alla seconda istanza. Notare che per informix il log potrebbe essere disattivato e questo renderebbe inusabile HDR.

La macchina di *stand by* è una macchina che ha lo stesso database della macchina principale, ma viene "foraggiata" con i *log* prodotti dall'altra. Questa seconda macchina macchina, nel caso la prima si guastasse, può molto facilmente prendere il posto di quella guasta e fornire lo stesso servizio in poco tempo. Spesso si fa in modo da mantenere un distacco temporale di almeno mezzora tra le due macchine perché così è anche possibile collegarsi direttamente alla seconda per recuperare dei dati erroneamente cancellati.

4 Conclusioni

Ci sono molti tipi di *cluster* open source che possono gestire tutte le principali configurazioni utili nelle aziende private e pubbliche italiane. Alcuni strumenti, `rsync` o `heartbeat`, e alcune applicazioni come `postgresql` o `JBoss`, spesso associati a GNU/Linux, sono in realtà molto utilizzati anche in ambiente Unix proprietari.

Vista la complessità del *cluster* è sempre meglio fare un'analisi completa degli scenari nei quali esso dovrà agire, prima di avventurarsi in acquisti e configurazioni. L'errore che viene fatto più spesso è proprio quello di sopravvalutare l'importanza del servizio da inserire nel *cluster* e disegnare un'architettura eccessivamente costosa o inutilmente complessa. Inoltre, per tutte quelle applicazioni che prevedono la gestione della ridondanza, è necessario sfruttare i meccanismi dell'applicazione.