MySQL cluster: un database ad alta affidabilità

Con una serie di articoli, Raoul ci indicherà come ottimizzare la performance delle applicazioni web utilizzando nel modo migliore il database MySQL, soprattutto nella versione cluster

di Raoul Scarazzini

Cerchiamo di capire con questa serie di articoli come sia possibile impiegare il più famoso database utilizzato per applicazioni web in soluzioni di alta affidabilità.

MySQL: Solo un server SQL?

Quanti non hanno sentito parlare almeno una volta di MySQL? MySQL è il database più diffuso sul web per una svariata serie motivi, la velocità, la facilità d'impiego, la semplicità di accesso ed il supporto per i sistemi operativi più utilizzati. È inoltre un progetto Opensource che pur avendo alla base una reale azienda che deve produrre profitto, la svedese MySQL AB, distribuisce il proprio software liberamente.

Il database infatti è liberamente scaricabile dal sito ufficiale http://www.mysql.com nelle versioni per i sistemi operativi Linux, Solaris, Mac OSX, FreeBSD e tantissimi altri. Un elenco completo delle piattaforme supportate si trova a questo indirizzo: http://dev.mysql.com/downloads/mysql/5.0.html

Detto questo, quello che molti non sanno è che MySQL AB sta sviluppando in parallelo al motore database standard anche una versione cluster, quello che viene definito un database ad alta affidabilità. Il progetto originale partì dalla necessità di un'importante azienda di telefonia di avere un motore database ad alta affidabilità che permettesse ricerche estremamente rapide.

La struttura relativamente semplice degli archivi dell'azienda si coniugava alla perfezione con le funzionalità offerte da MySQL, a cui mancava soltanto il supporto per l'alta affidabilità. Il prodotto finale di questi studi fu appunto MySQL cluster che da allora MySQL AB continua a far crescere in parallelo al motore standard.

Cercheremo di capire nel corso di questo articolo (e dei successivi) come funzioni e quali siano le potenzialità di questo progetto.

Cos'è un database ad alta affidabilità?

Un database ad alta affidabilità (o HA, High Available) è semplicemente quello che dice di essere: una base dati sempre disponibile, che presenta ridondanza di processi e di dati, che non ha cioè singoli punti di rottura (o SPOF, Single Point Of Failure).

All'interno di un'architettura senza SPOF tutte le componenti presenti (Hardware e Software) sono ridondate. Se una componente smette di funzionare, l'integrità del sistema non è compromessa.

Tale concetto viene applicato a tutti i tipici cluster di servizi, come nel caso di uno share di rete NFS o un Web server debbano essere ridondati al fine di garantire che, in caso di interruzione dei processi principali, i servizi continuino ad essere erogati.

In un database, avere ridondanza di processi significa che se per una qualsiasi ragione, il processo Master (ossia quello a cui le applicazioni client fanno solitamente riferimento) dovesse cadere, il database rimarrebbe comunque accessibile (grazie ad un secondo processo, che diventerebbe a sua volta Master).

Avere ridondanza di dati invece, significa che il dato non è registrato in una singola posizione. Quindi se questo si corrompe si ha la possibilità di ripristinarne il contenuto in base alla sua copia che risiede in una diversa posizione.

Tutte queste funzionalità permettono di avere un database senza SPOF: se si dovesse danneggiare una qualsiasi componente, logica o fisica, il sistema non risulterebbe compromesso.

Come ottenere un database ad alta affidabilità?

Definito quindi il concetto di database HA, va capito nella realtà come sia possibile ottenerne uno, ovvero come si possano eliminare tutti i potenziali SPOF.

Il discorso SPOF non va sottovalutato, ma è necessario porsi dei limiti nella considerazione di tutti i possibili punti di rottura. Questo significa che se una singola macchina risulta essere un unico SPOF, ed è quindi necessario averne due, allora anche la stanza in cui saranno presenti le due macchine sarà uno SPOF e quindi sarebbe bene separare in stanze diverse le due macchine, ma a quel punto l'edificio stesso diventerebbe uno SPOF e proseguendo di questo passo si arriverebbe ben presto a capire come l'universo stesso sia un unico immenso SPOF!

Quindi, senza tirare in ballo la meccanica quantistica per creare un database cluster, è chiaro come sia necessario trovare il giusto equilibrio tra affidabilità e funzionalità.

Separare su macchine diverse le componenti del database è un buon compromesso: i processi, ossia la parte software che permette l'accesso al dato ed i dati stessi saranno su macchine distinte.

Questa logica, divide il cluster in due livelli: quello SQL, inerente ai processi, e quello Storage, inerente ai dati. Entrambi i livelli devono essere esenti da SPOF e devono quindi prevedere ridondanza.

La ridondanza di processi (livello SQL)

È necessario che il motore SQL che gestisce le richieste di accesso e le operazioni di lettura/scrittura sia sempre disponibile. Ad un motore SQL generalmente corrisponde un processo in ascolto presso un determinato indirizzo IP su una particolare porta. Rendere questa situazione ridondata, significa avere più di un motore SQL. Ciascuno di questi motori, idealmente, dovrebbe essere posto dietro ad un bilanciatore, cioè un software che renda disponibile un solo indirizzo IP di riferimento ma che distribuisca il carico delle richieste effettuate attraverso tutti i motori che sono stati configurati. Esistono diversi software adatti a questo scopo, ma una discussione in merito andrebbe oltre gli scopi del presente articolo. Per il momento, si può concludere dicendo che questa situazione eliminerebbe lo SPOF legato al motore SQL.

La ridondanza dei dati (livello Storage)

Al pari della ridondanza dei processi, che riguarda più che altro l'alta affidabilità dell'accesso alla base dati, c'è la ridondanza dei dati stessi, che si può ottenere sostanzialmente in due modi o con una combinazione di entrambi<: la ridondanza fisica e la ridondanza logica.

Per ottenere ridondanza fisica è sufficiente avere hardware dedicato a questo scopo, ad esempio controller RAID che gestiscano la duplicazione dei dati su due o più dischi in maniera trasparente. In caso di rottura fisica del disco, questa viene segnalata in modo da poter essere corretta (con la sostituzione del disco), ma l'accesso ai dati rimane comunque garantito.

Un po' più complicato è il discorso relativo alla ridondanza logica. Per capire questo concetto, è necessario comprendere il significato di replica. Il numero di repliche all'interno di un database indica il numero copie multiple dello stesso dato. Le operazioni effettuate sul dato sono replicate su tutte le copie e fino a che questa operazione non è completata, la transazione che ha generato la modifica non è considerata conclusa. È chiaro quindi come esistano repliche primarie (le prime ad essere utilizzate) e repliche secondarie (le copie effettive) ed è altrettanto logico che avere due repliche del medesimo dato sullo stesso sistema non rispetti il principio dell'assenza di SPOF.

La soluzione ideale quindi sarebbe quella di avere come minimo due repliche distribuite su due diversi sistemi, come in figura 1.

Qui l'intero database è diviso in due partizioni. Nel primo sistema abbiamo la replica primaria della prima partizione e la replica secondaria della seconda partizione, mentre nel secondo, viceversa, è presente la replica primaria della seconda partizione e quella secondaria della prima. In questo modo in caso di rottura, blocco o qualsiasi evento comprometta il funzionamento di uno dei due sistemi, l'intera base dati è ricostruibile e l'accesso ai dati sempre disponibile.

La situazione ideale quindi, prevede che esista una base dati distribuita su almeno due sistemi, ciascuno dei quali contenente la propria replica primaria e la secondaria dell'altro sistema. L'accesso ai dati sarà reso disponibile da due motori Sql attraverso un unico indirizzo IP, gestito dal bilanciatore che distribuirà il carico delle richieste. In tutta questa struttura esiste un solo SPOF che è rappresentato dal bilanciatore stesso, che quindi a sua volta andrebbe ridondato.

Questa è la situazione perfetta (forse paranoica!) ed è illustrata in figura 2.

MySQL cluster

Definita quindi la struttura generale dei database cluster, possiamo spostare la nostra attenzione su come è composto un cluster MySQL.

Come tutti i database cluster, anche quello MySQL presenta la consueta architettura a due livelli: il livello SQL ed il livello Storage, denominato anche NDB Cluster, dove l'acronimo NDB sta per Network DataBase.

Ciascuna componente presente in un livello, è definita Nodo. Esisteranno quindi nodi client, ossia componenti appartenenti al livello SQL e nodi dati, cioè componenti del livello storage. Esiste anche una terza tipologia di nodo, denominato di management (gestione) che rappresenta il ful-

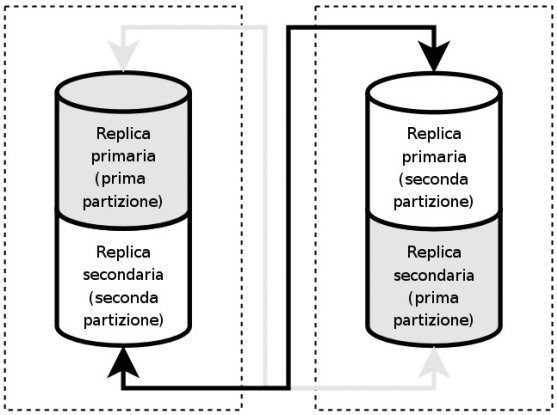

Figura 1 - La distribuzione di due partizioni, replicate una volta, su due sistemi

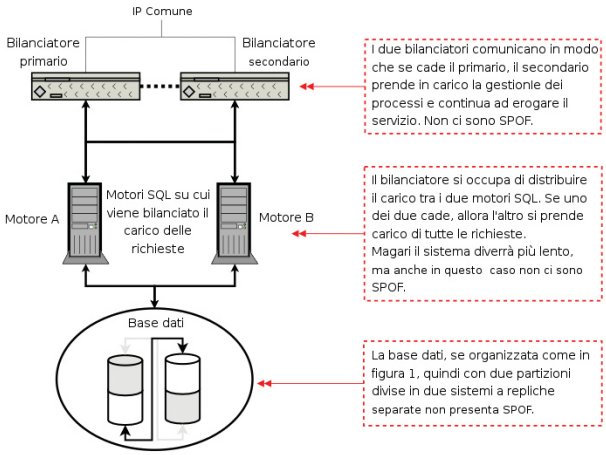

Figura 2 - La struttura logica di un database cluster che non possiede alcuno SPOF

cro intorno al quale ruota il MySQL cluster e sul quale verrà fatto un discorso a parte.

I nodi client

I nodi client sono ciò che permettono alle utenze di effettuare operazioni sui dati e si dividono essenzialmente in tre categorie:

1) Istanze dei server MySQL : la maggioranza dei casi, veri e propri server MySQL che si connettono alla base dati e si interfacciano all'esterno attraverso i classici processi demone mysqld.
2) Applicazioni esterne che utilizzano l'accesso nativo ai dati: si tratta di applicazioni che non usano l'intermediario MySQL per accedere al database, ma lo fanno direttamente utilizzando quelle che vengono definite NDB API. Questi client effettuano l'accesso ai dati direttamente dal codice del programma costruito.
3) Client nativi del cluster MySQL: ossia le applicazioni stesse del cluster, programmi cioè che consentono l'accesso e la manutenzione del cluster attraverso ad esempio operazioni di backup o restore. Tali applicazioni sono denominate NDB clients.

Tutti e tre i casi illustrati consentono all'utente di avere un interfaccia di accesso ai dati, in alcuni casi "consueta" (Il tipico server MySQL) ed in altri meno (le NDB API, delle quali bisogna avere una conoscenza piuttosto approfondita della programmazione database a basso livello).

I nodi dati

I nodi dati sono la struttura che si occupa della memorizzazione effettiva dei dati e compongono il già citato NDB Cluster.

Tutta quest'area lavora in maniera trasparente rispetto ai client. Il modo e la posizione in cui i dati vengono frammentati e la gestione delle repliche non sono conosciute dai client. Essi accedono, con i metodi illustrati poco sopra, solo all'entità NDB cluster. A ciascuno dei nodi dati è associato un processo demone denominato ndbd.

Una cosa importante da capire è la sostanziale differenza di NDB cluster dagli altri prodotti database cluster in commercio, come ad esempio Oracle RAC. In questi infatti, l'alta affidabilità è garantita da una totale ridondanza. Questo significa che esistono due macchine identiche, ma una sola è attiva e si preoccupa di tenere allineata la seconda che rimane pronta a subentrare in caso di guasto.

All'interno di NDB cluster, tutti i nodi che fanno parte del sistema pur rispettando le regole di ridondanza e assenza di SPOF, contribuiscono al funzionamento del cluster incrementandone notevolmente le prestazioni.

Il nodo management

La parte più importante del cluster MySQL è rappresentata dal nodo di management, il nodo al quale tutte le componenti del cluster si connettono appena avviate per conoscere il resto del sistema.

In questo nodo è configurata la struttura del cluster: quanti e quali sono i nodi dati, quante repliche avranno le partizioni del sistema, quanta memoria sarà disponibile e così via.

Dal nodo di management è possibile avviare e stoppare i nodi dati, verificare lo stato del loro funzionamento, effettuare backup e restore, settare la tipologia dei log di ciascun client e stoppare l'intero cluster.

É interessante notare che, una volta che il cluster sia stato avviato, il nodo di management potrebbe tranquillamente smettere di funzionare. Tutti i nodi del cluster infatti, una volta ricevuti i parametri di configurazione dal nodo di management, iniziano a comunicare direttamente fra loro. Certo è che se si dovesse interrompere il processo del nodo di management sarebbe impossibile effettuare le altre operazioni descritte sopra, ma è importante capire che il cluster stesso, una volta che sia completamente avviato, sia indipendente dal nodo di management.

Il nodo di management è un processo demone, avviato tramite un file eseguibile chiamato ndb_mgmd che rimane in ascolto su una porta alle richieste dei nodi client. In figura 3 viene illustrata la struttura logica del cluster MySQL.

Replicazione e frammentazione dei dati nel cluster MySQL

Ora che abbiamo capito la logica del cluster MySQL cerchiamo di concentrarci su come esso tratti la distribuzione dei dati. Questa è divisa in due parti: la frammentazione dei dati e la replicazione degli stessi.

La frammentazione dei dati consiste nel partizionare una tabella in un numero pari di frammenti che vengono divisi all'interno di un numero pari di nodi. La replicazione consiste nella duplicazione dei frammenti all'interno dei nodi.

Osserviamo la figura 4: in questo esempio, il database è organizzato in due nodi e due repliche. Tutto si rifà alla teoria dei cluster di database che abbiamo precedentemente spiegato ed illustrato in figura1. Qui ciò che veniva chiamato partizione è chiamato frammento ma il concetto non cambia: ciascun frammento di tabella è presente su entrambi i nodi, in un nodo come frammento primario e nell'altro come replica di questo.

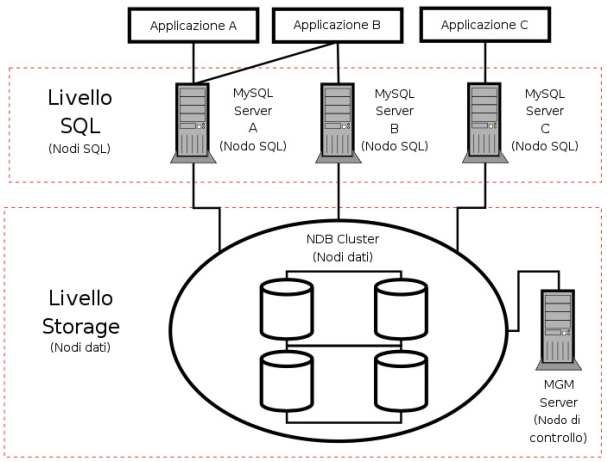

Figura 3 - La struttura logica del cluster MySQL

Il danneggiamento di un nodo non compromette l'intero sistema in quanto ogni sistema possiede le informazioni necessarie a ricostruire il totale delle informazioni presente nel sistema.

Si noterà come nella figura venga menzionato il "Nodegroup". All'interno del cluster MySQL il numero di Nodegroup si ottiene con questa semplice equazione: Numero di Nodegroup = Numero di data nodes / Numero di repliche

Nel caso di figura4 quindi con due nodi dati e due repliche avremo un solo Nodegroup. Se invece avessimo a disposizione quattro macchine, potremmo organizzare il carico come illustrato in figura5, dove esistono 4 nodi dati con due repliche e quindi due NodeGroups.

Le tabelle in questo esempio sono frammentate in quattro partizioni e sono divise a coppie nei Nodegroup: ciascun Nodegroup contiene metà delle informazioni del database e ciascun nodo dati possiede un frammento primario dei quattro in cui è divisa la tabella.

Anche questa struttura non possiede SPOF. Le combinazioni possibili del numero di repliche, dati e nodi non devono superare i limiti del cluster MySQL che sono di un massimo di: 4 repliche dei dati, 48 nodi dati, 64 nodi totali. Lavorando per coerenza con un numero pari di repliche e nodi dati, si possono ottenere situazioni che partono da due nodi con due repliche fino a quarantotto nodi con quattro repliche.

É chiaro che maggiore sarà il numero di repliche, minori saranno le performance in quanto le transazioni di scrittura non saranno completate sino a che l'informazione non verrà scritta in tutte le sue repliche.

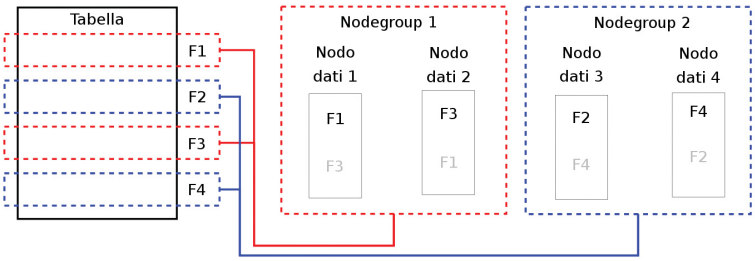

Figura 5 - Le tabelle del DB sono partizionate verticalmente in quattro frammenti. Ciascun node group contiene metà delle informazioni dell'intero database

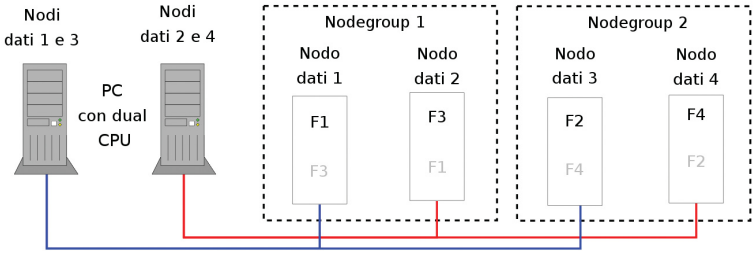

Figura 6 - Ciascun PC con doppia CPU contiene la metà di un nodegroup. Con questa configurazione se una delle due macchine va in crash, il sistema non risulta compromesso

Come organizzare CPU e Memoria sui nodi dati

L'organizzazione delle risorse fisiche del database è abbastanza semplice, nel senso che esistono delle regole precise su cui basarsi.

CPU

Ogni nodo dati deve essere associato ad una CPU, quindi è perciò sconsigliato configurare un sistema a singola CPU affinché esegua più di un processo ndbd (ossia il processo nodo dati del cluster). Per eseguire più di un nodo dati su una singola macchina è necessario usare sistemi Multi CPU, ma bisogna ricordarsi di eliminare lo SPOF macchina singola, pertanto, è necessario predisporre un minimo di due macchine con dual CPU.

Anche dal punto di vista logico, nel momento in cui si decide di mettere sulla stessa macchina due nodi data, è bene separare le risorse in modo che se una macchina va in crash il sistema non risulta compromesso: quindi nel caso in cui si decida di avere un database con quattro nodi data distribuiti su due macchine dual CPU sarà bene organizzare i Nodegroup come in figura 6: con questa organizzazione, se anche una macchina va in crash, il sistema non è compromesso.

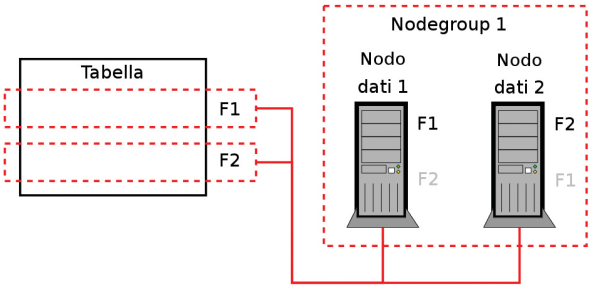

Figura 4 - Le tabelle del DB sono partizionate verticalmente in due frammenti. Ciascun frammento è replicato una volta in un nodo

Memoria

La quantità di memoria disponibile nel cluster, viene calcolata con questa semplice equazione: Memoria totale cluster = Memoria RAM Nodo dati * Numero totale di nodi / Numero di repliche

Quindi se avremo quattro gigabyte di memoria su ogni nodo dati, quattro nodi dati totali e due repliche, il nostro cluster avrà otto gibabyte di memoria disponibile.

Conclusioni

Abbiamo visto in questo articolo come è strutturato logicamente e come funziona il cluster MySQL. Nel prossimo eseguiremo una prova effettiva, configurando un cluster a due nodi ed imparando a gestire il tutto dal Management server.

Raoul Scarazzini è resonsabile del settore Linux presso Cutaway SAS, una società di consulenza IT a Milano.