mokabyte.it

Il teorema CAP... in Brewer - MokaByte

Luca Vetti Tagliati

16-22 minuti

In questo terzo articolo della serie dedicata al teorema CAP o di Brewer, concludiamo la trattazione su Oracle Coherence, entrando in particolari tecnici della gestione del grid e mostrando nei dettagli come la scelta operata di continua disponibilità e consistenza (AC) sia stata ottenuta, giocoforza, a discapito della tolleranza alle partizioni. Nei prossimi numeri, comunque, continueremo il discorso sul principio di Brewer mostrando invece una soluzione basata su un compromesso di tipo AP.

Introduzione

Negli articoli precedenti abbiamo illustrato le caratteristiche fondamentali di Oracle Coherence a partire dalla sua definizione: “a continuously available in-memory data grid across a large cluster of computers for real-time data analysis, in-memory grid computations and high-performance processing solution” (data-grid internamente gestito in memoria, sempre disponibile grazie a un elevato cluster di computer, adatto per l’analisi dei dati in tempo reale, calcoli di tipo grid eseguiti in memoria, e soluzioni ad alte prestazioni).

In questo articolo entriamo in **particolari tecnici** relativi alla gestione del cluster illustrando in dettaglio sia i **protocolli** di **comunicazione**, sia i **protocolli applicativi** per la gestione del **cluster**. Ciò permetterà di comprendere a fondo il **compromesso AC** e il temuto scenario del cosiddetto **split brain** ("cervello diviso"), ossia la creazione di due sotto-cluster autonomi, che, in situazioni limite, può causare la perdita di dati.

Indipendentemente dall'argomento trattato, questo articolo dovrebbe rappresentare un'interessante lettura per tutti coloro che si occupano di architettura dei sistemi, poiche' mostra alcune interessanti soluzioni progettuali. Inoltre dovrebbe dimostrare ancora una volta come anche la conoscenza dei dettagli tecnici sia precondizione imprescindibile per riuscire a disegnare architetture che funzionano.

TCMP Tangosol Cluster Management Protocol

Il cluster Coherence è formato da una serie di membri che comunicano tra loro per mezzo di un sofisticato protocollo denominato **TCMP** (**Tangosol Cluster Management Protocol**, Protocollo di Gestione Cluster Tangosol, dove Tangosol è il nome dell'azienda che ha progettato e implementato Coherence prima dell'acquisizione da parte di Oracle). Per la precisione, è più corretto esprimersi in termini di una **suite di protocolli** basati (ovviamente) su **IP** (Internet Protocol).

I servizi implementati

TCMP è utilizzato per implementare tutti i servizi richiesti per la gestione del grid [1], tra i quali i più importanti sono:

- la **scoperta** dei membri del cluster (che in questo contesto è rappresentato da un insieme di **JVM**);
- la **gestione** del cluster stesso;
- i servizi di **provisioning**;
- la **trasmissione** dati.

L'**IP** [2], molto brevemente, è un **protocollo** di livello di **network** (**OSI Layer 3**, figura 1) che contiene informazioni di indirizzamento e alcune informazioni di controllo necessarie per consentire ai pacchetti di essere instradati. IP è parte della RFC 791 ed è il protocollo di livello di rete primario nello stack di protocolli Internet: **IP insieme con TCP**, rappresenta il cuore dei protocolli Internet. IP ha due compiti fondamentali:

- fornire connessione, consegna best-effort di datagrammi attraverso un internet work;
- fornire la frammentazione e il riassemblaggio dei datagrammi.

Figura 1 – Il modello OSI e la suite dei protocolli Internet.

I cluster

Un cluster contiene i servizi condivisi da tutti i membri del cluster: servizi di **connettività** (come per esempio il Cluster Service), i servizi di **cache** (come ad esempio il servizio di cache distribuita),

e servizi di **elaborazione** (come ad esempio il servizio di invocazione). Ciascun membro del cluster è in grado di fornire e utilizzare tali servizi. Il primo membro del cluster viene indicato come **senior member** ("membro anziano") e si occupa di iniziare i servizi di base necessari per creare il cluster. Se il **senior member** del gruppo viene fermato o diviene non disponibile, un altro membro del cluster assume il ruolo di membro anziano.

I protocolli

Vediamo ora le varie combinazioni di protocolli utilizzati da TCMP.

UDP/IP Multicast

UDP/IP **multicast** ha tre utilizzi principali.

Anzitutto serve per la **scoperta cluster**: lo utilizzano i candidati membri del cluster per scoprire se esiste un cluster in esecuzione a cui iscriversi.

Vi è poi il concetto di **heartbeat** ("battito cardiaco"): il senior member del gruppo ha la responsabilità, con cadenza ben definita, di emettere un "battito cardiaco" tramite **UDP Multicast**. L'intervallo di tempo tra due battiti è configurabile. Il valore di default è un secondo. La tecnica del "battito cardiaco" è utilizzata per assicurarsi che i vari nodi siano raggiungibili e operativi. Quindi è un servizio fondamentale utilizzato per assicurarsi il corretto funzionamento del cluster

Infine c'è la **consegna messaggi**: i messaggi che devono essere consegnati contemporaneamente a diversi membri del cluster sono tipicamente inviati tramite **multicast**; **unicasting** è invece preferito quando il messaggio ha un solo destinatario.

Da notare che UDP multicast è il protocollo ideale per l'implementazione dello heartbeat, in quanto un "battito cardiaco" è, per sua natura, una comunicazione leggera, non orientata alla connessione, destinata a diversi membri.

UDP/IP Unicast

UDP/IP **unicast** è invece utilizzato per due usi principali.

Come primo aspetto, ci sono le **comunicazioni point-to-point** (comunicazione diretta tra due membri del cluster). Questa tipologia di comunicazione è necessaria per scambio di messaggi, riconoscimenti asincroni (**ACK**), riconoscimenti negativi asincroni (**NACKs**) e "battiti cardiaci" **peer-to-peer**.

Come seconda casistica, in circostanze particolari, Unicast può essere usato anche per **messaggi diretti** a più membri del cluster. Questa strategia è utilizzata per modellare il flusso di traffico e per contenere il carico della CPU in cluster di grandi dimensioni.

Figura 2 – Rappresentazione logica di un invio Multicast.

TCP/IP

TCP/IP è utilizzato come "anello" opzionale necessario per gestire casi particolari in cui l'affidabilità intrinseca è caratteristica

fondamentale come per il servizio di “rilevamento morte”. In questo caso il ricorso all’anello TCP è molto conveniente per poter distinguere in maniera affidabile tra una situazione di errore del nodo e una in cui un nodo non reagisce come atteso o nei limiti previsti, magari perche’ il nodo (la JVM) sta eseguendo un GC completo.

Da notare che il protocollo **TCP/IP non è** mai **usato** come un **meccanismo** di **trasferimento dati** a causa della sua intrinseca “inefficienza” dovuta sia al necessario overhead sia alla natura bloccante della comunicazione richiesta dalle comunicazioni sincrone.

Considerazioni sui protocolli

Per essere precisi, non è corretto esprimersi in termini di inefficienza. Il protocollo **TCP/IP** (livello 4 di trasporto, cfr. figura 1) offre tutta una serie di meccanismi supplementari come orientamento alla connessione, elevata affidabilità grazie ai servizi di gestione del flusso, della congestione, etc. servizi che finiscono inevitabilmente per avere un’incidenza sulla latenza di trasmissione. Il meccanismo di affidabilità del protocollo TCP, per esempio, consente a dispositivi connessi di risolvere problemi relativi alla perdita, ritrasmissione, alla consegna in ritardato e alla duplicazione di pacchetti. Tuttavia, questi meccanismi hanno un **impatto sulle performance**, impatto trascurabile in un numero elevato di scenari, ma tipicamente non sostenibile in sistemi dove la latenza gioca un ruolo centrale.

Di contro, il protocollo **UDP** è molto leggero, ad altre prestazioni e, in quanto asincrono, non orientato alla connessione, privo di tutta una serie di meccanismi come l’elevata “affidabilità”, etc. Il suo

utilizzo in ambienti come i grid di Coherence, in cui la consistenza è caratteristica fondamentale, richiede, quindi, l'implementazione esplicita di un layer supplementare necessario per assicurare la consegna affidabile dei messaggi.

La scelta operata da Oracle Coherence, in maniera assolutamente compatibile con altre tecnologie operanti nello stesso spazio (come per esempio **JGroups**), consiste nell'utilizzare il protocollo **UDP**, e di **implementare esplicitamente** i **servizi** di **affidabilità** in maniera fortemente ottimizzata allo specifico dominio di impiego (gestione di un grid). **TCMP** infatti implementa un protocollo completamente asincrono di **ACK-** e **NACK**, per garantire sia l'affidabilità del protocollo, sia l'ordinamento dei messaggi. In particolare, il protocollo TCMP, per ogni pacchetto spedito via UDP, si attende un esplicito **acknowledgement** (**ACK**). Per questioni di efficienza, i pacchetti non sono tipicamente "confermati" singolarmente ma a gruppi e quindi il protocollo gestisce una coda di **ACK/NACK**. Qualora un pacchetto non venga confermato nel lasso di tempo previsto (**timeout**), il nodo mittente effettua il log del pacchetto; ciò fa sì che i nodi membri del cluster inizino una consultazione per capire la natura del problema. L'algoritmo utilizzato è denominato "del testimone" (**witness protocol**, descritto nel paragrafo successivo) e può portare alla rimozione di uno o più nodi dal cluster.

L'esteso utilizzo dei **protocolli asincroni** è un principio fondamentale del cluster Coherence. Questi sono utilizzati per far sì che le comunicazioni non siano bloccanti neanche quando molti thread in esecuzione su un medesimo nodo comunicano contemporaneamente. Comunicazione asincrona, tra l'altro, implica che la latenza della rete non influenza il throughput del

cluster, benche', ovviamente, abbia un'influenza diretta sulla velocità con cui sono espletate diverse operazioni.

Nonostante i benefici apportati dalla comunicazione **UDP Multicast** in tutta una serie di scenari (come per esempio nell'implementazione del "battito cardiaco", per l'invio di dati a diversi nodi), è importante notare che non sempre questa trasmissione è permessa dalla rete, anzi… Nella maggior parte degli ambienti **WAN**, e alcuni ambienti **LAN**, questa tipologia di **traffico non** è **consentita** e viene tagliata dai vari dispositivi di rete.

In queste circostanze, Coherence può essere configurato per non utilizzare comunicazioni **UDP Multicast**, rimpiazzandole con **UDP Unicast**. Da tener presente che raramente è una buona idea gestire un cluster distribuito globalmente e quindi attraverso una WAN. Qualora ciò sia realmente necessario, è verosimilmente più opportuno pensare a una serie di cluster indipendenti interconnessi: federazione di cluster.

Coherence è stato progettato per utilizzare il più possibile **comunicazioni dirette** tra i nodi (**point-to-point**) per lo scambio dei dati (**Share-Nothing architecture**, architettura a zero condivisione), quindi, qualora si sostituisca l'Unicast al Multicast, la maggior parte dei servizi non subisce grandi perdite di performance. Nondimeno, esiste tutta una serie di servizi in cui il **Multicast** è la soluzione ideale e permette di raggiungere performance superiori. Gli esempi più lampanti sono: servizio di scoperta del cluster, heartbeat e consegna del medesimo messaggio a diversi nodi (situazione tipica in cui si configura il cluster per avere diversi nodi di backup). Questi servizi, se realizzati attraverso UDP Unicast, subiscono una notevole (in termini di low-latency) penalizzazione dalla mancanza della

comunicazione Multicast.

Thread e protocolli

Un elemento importante per la scalabilità di Coherence è il **ridotto** e **costante** utilizzo di **thread** dedicati alla **comunicazione**. In particolare, il protocollo TCMP funziona con **solo tre** socket **UDP/IP** (uno multicast e due unicast) e **sei thread** per nodo (**JVM**), indipendentemente dalla dimensione del cluster. Questo è un elemento chiave per la scalabilità di Coherence: indipendentemente dal numero di server, ogni nodo del cluster può comunicare sia **point-to-point** o con **insiemi** di membri del cluster senza la necessità di ulteriori connessioni di rete. Inoltre, qualora l'anello di TCP/IP sia utilizzato, allora TCMP utilizza alcuni socket TCP/IP aggiuntivi gestiti da un unico thread aggiuntivo.

Il protocollo del testimone (witness)

Come visto in precedenza, Coherence utilizza un meccanismo sofisticato ed intelligente per la rilevazione del fallimento dei nodi basato sullo **heartbeat** e sul **protocollo del testimone**. Il compito del meccanismo di **heartbeat** è di assicurarsi che tutti i nodi del cluster siano operativi, o, con visione pessimistica, di individuare nodi non più funzionanti per avviare la procedura di espulsione del nodo dal cluster.

In particolare, qualora un nodo fallisca a rispondere, il nodo mittente effettua il log che fa sì che i nodi membri del cluster inizino una consultazione per capire la natura del problema (figura 3). Infatti, prima di poter decretare la morte di un nodo "sospetto", due altri nodi, selezionati volutamente in modo casuale, sono chiamati ad agire da "testimone" (da qui il nome dell'algoritmo). Se

entrambi i nodi confermano che il nodo sospetto non risponde, allora viene ufficialmente dichiarato morto e quindi espulso dal cluster. D'altra parte, se i nodi testimoni non concordano unanimemente, allora il nodo "accusatore" viene rimosso.

Figura 3 – Schematizzazione del "protocollo del testimone".

Come visto nell'articolo precedente, una volta che un nodo è stato rimosso dal cluster, gli altri nodi si fanno carico della porzione dei dati da esso memorizzato. In particolare, avviene la **ridistribuzione** di tutti i suoi dati sia per assicurare un backup completo, sia per mantenere una distribuzione uniforme dei dati in tutto il cluster.

Split brain

Split brain ("cervello diviso") è uno scenario pericoloso originato dalla **disconnessione**, tipicamente temporanea, di **alcuni nodi** di un cluster che si **riorganizzano** in un cluster di dimensioni inferiori e iniziano a **operare in autonomia**. In particolare, ogni sottogruppo crede che i nodi appartenenti all'altro gruppo non siano più operativi e quindi li dichiara morti; inoltre, assumendo di essere membri dell'unica parte superstite del cluster, si riorganizzano e iniziano ad operare in autonomia. Questo scenario può aver luogo quando la connessione è relativamente lenta (ad esempio diversi millisecondi) e/o non affidabile,

condizioni facilmente verificabili quando si utilizza un modello di deployment non particolarmente brillante che prevede che i nodi del cluster siano distribuiti su due o più data center in configurazione **hot-hot** (figura 4). Con questa pessima strategia, la probabilità di dar luogo a due sottocluster, uno per centro, è abbastanza elevata.

Figura 4 – Pessima strategia di deployment/continua disponibilità.

Uno scenario **split brain** può causare la perdita di dati, perche' i dati possono essere aggiornati in modo differente sui due cluster, si possono avere duplicati di chiave primaria e alcuni aggiornamenti potrebbero andare persi durante la risoluzione dei sottocluster. Un'altra classica situazione in cui questo scenario può portare alla perdita dei dati è con un funzionamento in **write-behind**.

Sebbene in architetture ben studiate e analizzate, lo scenario dello **split brain** non sia assolutamente comune, e nonostante che Coherence abbia sviluppato una serie di sofisticati meccanismi sia per individuare l'occorrenza di questa condizione sia per intraprendere opportune procedure di risoluzione, la **tolleranza alle partizioni** rimane (da progetto) il punto debole del grid Coherence: si tratta della la caratteristica sacrificata in favore della continua disponibilità e completa consistenza dei dati.

Panic Protocol

Qualora un membro del cluster coherence si renda conto della la presenza di più (sotto-) cluster, ossia qualora si renda conto che si è creata una situazione di **split brain**, Coherence esegue un opportuno protocollo denominato “di panico” al fine di risolvere i cluster in conflitto e **consolidare** tutti i membri in un unico cluster. Il principio di funzionamento del protocollo consiste nella rimozione dei cluster più piccoli fino a quando c’è un unico cluster rimanente. Nel caso di gruppi di uguali dimensioni (in termini di nodi), sopravvive quello in cui risiede il membro più anziano.

Conclusioni.

Con questo articolo abbiamo terminato la mini serie dedicata al grid Oracle Coherence analizzato dal punto di vista del teorema **CAP** / **Brewer**. In particolare, per comprendere appieno le scelte di disegno e la preferenza di orientarsi verso un compromesso di tipo **AC** (**continua disponibilità** e **alta consistenza**), abbiamo analizzato i protocolli di funzionamento di Coherence con particolare riferimento all’algoritmo del testimone. Come visto, in Coherence lo scenario pericoloso è dato dalla situazione di partizionamento del cluster (split brain): i nodi si organizzano in sottocluster autonomi, credendo che i nodi degli altri, non più raggiungibili, siano morti. Questa configurazione è pericolosa in quanto può portare alla perdita di dati.

Sebbene, in architetture ben studiate e analizzate lo scenario dello split brain non sia frequente la tolleranza alle partizioni rimane il suo punto debole, sacrificato in favore della continua disponibilità e completa consistenza dei dati.

Riferimenti

[1] Rob Misek – Jon Purdy, “Coherence Knowledge Base: Network Protocols”

http://coherence.oracle.com/display/COH31UG/Network+Protocols

[2] Cisco, “Internetworking Technology Overview: Internet Protocols”, June 1999

http://fab.cba.mit.edu/classes/MIT/961.04/people/neil/ip.pdf