File System Distribuito ad Alta Disponibilità

di Luca Paganelli
C.d.L.S. in Ingegneria Informatica
Corso di Reti di Calcolatori LS
AA 2003/2004

1 Studio del Problema

In questa primissima sezione si cercano di dettagliare gli scopi fondamentali del progetto motivando in linea generale le scelte esposte nelle successive sezioni.

Il requisito di base di un servizio di file system è dato dalla possibilità di organizzare contenuti digitali secondo una struttura gerarchica con la garanzia di poter riaccedere in qualunque momento ai medesimi. Tale garanzia implica requisiti molto forti di tolleranza ai guasti che non tutti i prodotti attuali sono in grado di soddisfare.

Le soluzioni locali più popolari - integrate ovviamente nei sistemi operativi proprietari - sono in grado infatti di raggiungere un livello di robustezza sufficiente ad applicazioni poco strategiche in quanto soffrono tanto dei problemi di carattere hardware quanto dei problemi software (in particolare dei crash del sistema operativo).

Per quanto riguarda i guasti in hardware esiste già una tecnologia molto collaudata, il R.A.I.D., che però può far fronte ai soli fault del sottosistema disco della macchina. In generale R.A.I.D. è in grado di scongiurare il pericolo della perdita definitiva dei contenuti ma rimane pur sempre soggetto a guasti su altre componenti. In altri termini R.A.I.D. non soddisfa il requisito di disponibilità perchè può sperimentare temporanee *denial of service*.

Un servizio di file system *distribuito* consiste invece nella possibilità di garantire l'accesso a file e directory da un qualunque nodo della rete che interconnette un sistema di macchine. Indipendentemente dall'organizzazione e dall'implementazione del servizio medesimo ogni cliente deve poter aver accesso ai medesimi contenuti, in altri termini ogni client deve avere la medesima visione del file system e dovrebbe avere la *sensazione* di fruire di un servizio locale.

A differenza del semplice file system *remoto*, di cui un esempio eccellente è NFS e che consiste nella pubblicazione in rete di una fetta del file system locale che diverrà liberamente accessibile da qualsiasi client che esplicitamente si connetterà a questa specifica macchina, un servizio *distribuito* non prende corpo dall'attività di un singolo attore ma si manifesta come cooperazione di più entità interconnesse, i nodi.

Se R.A.I.D. da un lato rappresenta la soluzione più completa in fatto di affidabilità, il modello distribuito è la soluzione ottima per quanto concerne la disponibilità. L'idea è quindi di produrre un sistema che goda delle proprietà di queste due soluzioni, in particolare risulta necessario sviluppare un *sistema distribuito con replicazione*. Si sottolinea come un prodotto di questo tipo raggiunga un livello di affidabilità ben superiore a R.A.I.D. poichè può far fronte anche a condizioni particolarmente catastrofiche in cui uno o più nodi vengano fisicamente distrutti!

Possiamo a tutti gli effetti definire questa soluzione un *cluster ad alta disponibilità* ovverosia un sistema di nodi indipendenti che si coordinano per erogare un servizio che dev'essere sempre attivo.

2 Architettura Generale

In questa sezione si illustra brevemente l'architettura generale del sistema avvalendosi di un plausibile esempio (Fig. 1). Nelle sezioni successive si entrerà nel particolare delle diverse problematiche che questo tipo di architettura porta con se.

Il cluster nasce dunque dal coordinamento di diversi processi nodo che nell'esempio sono siglati con A, B, C e D selezionati da un riquadro grigio. Essi rappresentano le unità di replicazione del file system (quelli che per R.A.I.D. erano i dischi fissi) e la loro attività più importante consiste proprio nel mantenimento della versione aggiornata per ogni file.

Questa soluzione ricalca infatti il modello di replicazione a copie attive poichè il servizio è fruibile da uno qualsiasi dei nodi (rete peer-to-peer) ed implica un ingente lavoro di sincronizzazione delle copie che verrà dettagliatamente discusso nelle sezioni successive.

Oltre al canale di comunicazione di gruppo, ogni nodo dispone anche di due canali di comunicazione privata: il primo (evidenziato nel disegno in colore viola) è dedicato all'interconnessione diretta con un altro nodo del cluster ed è utilizzato nelle conversazioni che non necessariamente devono essere pubbliche al fine di non sovraccaricare inutilmente il canale di gruppo (ad esempio per il trasferimento di file); l'altro (in turchese) è invece destinato ad accogliere nuovi clienti utilizzatori e costiuisce, come meglio espresso di seguito, la porta principale per l'accesso al servizio.

L'interazione del client con il cluster è realizzata infatti mediante un agente intermedio che esegue sul client stesso: il *Client Proxy*. Esso costituisce

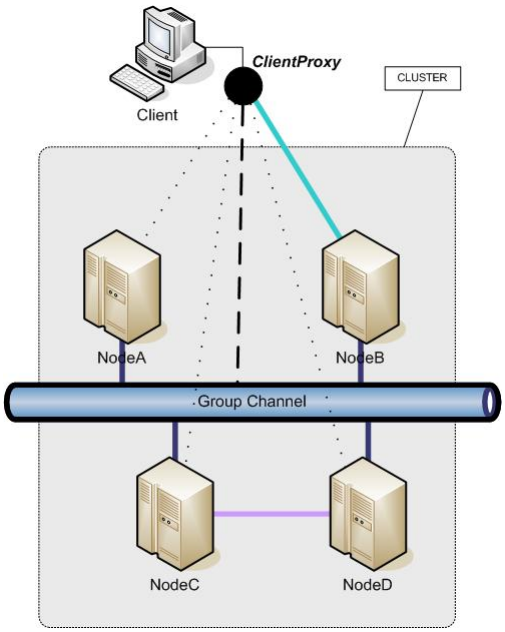

Figura 1: Architettura del Cluster

l'interfaccia di accesso al cluster realizzando la trasparenza di quest'ultimo all'allocazione ed all'organizzazione interna.

Client Proxy utilizza - linea tratteggiata grossa - il canale di gruppo per la *discovery* dei nodi che fanno parte del cluster e ne elegge uno arbitrariamente con cui instaura la connessione diretta (linea turchese) citata in precedenza. Da quel momento questo nodo diviene di fatto il fornitore del servizio per lo specifico client.

Nel caso in cui, per svariati motivi, il nodo dovesse interrompere tale comunicazione il Client Proxy si incaricherà in maniera completamente automatica della migrazione su un altro nodo del cluster (attivando una delle linee a puntini) e proseguirà le attività sul nuovo in maniera completamente trasparente al client.

In questo modo il client *vero e proprio* si trova ad interagire in maniera propriamente diretta solamente con il Client Proxy a cui espone dunque le proprie esigenze e da cui riceve le opportune risposte che l'agente avrà generato sulla base della reale interazione che sarà stata sostenuta con il cluster.

Client Proxy rappresenta in un certo senso un'evoluzione del *v*-node di NFS e, volendo accettare quest'ardito paragone, l'operazione di *mount* consiste nell'impostare le coordinate del canale di comunicazione piuttosto che l'indirizzo della specifica macchina che offre il servizio.

3 Definizione del Cluster

In questa sezione verranno analizzate tutte le problematiche relative alla corretta gestione del gruppo sia in fase di costituzione che in fase di manutenzione dinamica, nella sezione successiva si analizzeranno gli impatti delle richieste di servizio da parte dei client sull'attività del gruppo.

3.1 Gestione del Gruppo

Il primo requisito fondamentale in un sistema coordinato è un efficace modello di gestione del gruppo che consenta in ogni momento a tutti i nodi di pervenire alla medesima conoscenza sulla composizione del cluster e che tale conoscenza corrisponda effettivamente alla realtà.

Si analizzano di seguito i tre scenari più importanti di modifica dinamica della composizione evidenziando in che modo tale requisito sia soddisfatto.

3.1.1 Join

Per *Join* si vuole intendere l'evento in cui un nuovo nodo esprime il desiderio di entrare a far parte del cluster. Tale desiderio è espresso sul canale di gruppo ed ogni nodo è tenuto a fornire in risposta le proprie credenziali e la lista di tutti i nodi che compongono attualmente il cluster.

Si potrebbe pensare che questa informazione aggiuntiva sia assolutamente ridondante poichè in condizioni di corretto funzionamento al nuovo entrante perverrebbero le credenziali da tutti i nodi e avrebbe comunque la possibilità di dedurre la composizione del gruppo. In realtà è essenziale che il nodo abbia una qualche conoscenza di quante e quali risposte attendersi perchè altrimenti si rischierebbe o l'attesa infinita o comunque una conoscenza potenzialmente parziale.

Nel momento in cui, sulla base di quest'informazione e di tutte le risposte ottenute, stabilirà di aver ottenuto la composizione completa, il nodo affermerà il proprio ingresso nel cluster ed i componenti aggiorneranno la loro conoscenza.

È opportuno ivi specificare che benchè questa architettura preveda evidentemente un modello di coordinamento di tipo peer-to-peer (in cui nessuno cioè si accolla il compito di *master*) ad ogni nodo deve essere associato un qualche valore di priorità univoco nel gruppo per dirimere eventuali situazioni di collisione nei protocolli di mutua esclusione discussi in seguito. L'assegnazione dell'opportuno livello di priorità avviene ad opera del nuovo entrante medesimo sulla base del 'chi tardi arriva, male alloggia', ovverosia individuando il nodo componente con il massimo valore ed autoassegnandosi un valore superiore. Tale scelta viene ovviamente comunicata al gruppo nell'affermazione di ingresso.

3.1.2 Leave

La *Leave* di un nodo rappresenta l'evento dell'abbandono *graceful* del medesimo dal cluster il quale è preceduto da una segnalazione sul canale di gruppo dell'intenzione.

Tale segnalazione ha il solo obiettivo di velocizzare le operazioni di aggiornamento della conoscenza sulla composizione del gruppo ma bisogna specificare che nessun nodo ha la capacità di opporre veto all'abbandono.

3.1.3 Heart Beat

Per rendere il sistema robusto anche a variazioni improvvise del cluster (ad esempio nel caso in cui un nodo sperimenti un crash ed abbandoni forzatamente senza poter prima esprimere la leave) è stato previsto anche un protocollo di controllo di composizione che va sotto il nome di *Heart-Beat*.

In qualsiasi momento, e per qualsiasi ragione, un nodo può esprimere l'esigenza di controllare la composizione del gruppo attraverso una 'stimolazione' di heart beat sul canale comune.

La risposta di ogni nodo a tale stimolazione ha priorità assoluta su ogni altra attività in corso quando ricevuta. Nel caso in cui l'iniziatore non arrivasse a collezionare le risposte da tutti i nodi appartenenti alla sua conoscenza di composizione del cluster si prevede che ritenti l'operazione per un numero finito di volte fino a decretare che il gruppo ha mutato composizione.

In ogni caso al termine delle operazioni l'iniziatore emette un bollettino in cui afferma la nuova composizione del gruppo a cui ogni nodo dovrà attenersi. Un nodo che disgraziatamente dovesse ricevere un bollettino in cui non presenzia come componente sarà costretto a rieffettuare la *Join*.

Nel momento in cui si dovesse accertare la caduta di un nodo (o perchè iniziatori dell'HeartBeat o attraverso il bollettino) è necessario annullare tutti i lock che tale nodo aveva guadagnato.

3.2 Replicazione delle Risorse

Analizzate le più generali operazioni di gestione del gruppo vediamo nello specifico come un nodo appena entrato debba comportarsi per contribuire attivamente al soddisfacimento dei requisiti di affidabilità che il cluster deve soddisfare.

In un modello peer-to-peer a copie attive è necessario che ogni nodo sia in grado di fornire il medesimo servizio, nello specifico trattandosi di un servizio di file system questo significa che dev'essere garantito che ogni nodo possegga i contenuti più aggiornati del cluster (si eviti insomma che il medesimo file richiesto *contemporaneamente* su due nodi diversi differisca in contenuto).

Lasciando le problematiche dell'aggiornamento delle versioni a discorsi successivi si vuole tuttavia qui sottolineare che nel preciso momento in cui il nodo accede al gruppo, prima di potersi dire pronto ad erogare il servizio, è necessario che replichi su se stesso tutte le risorse (file) esposte dal cluster. Per fare ciò il nodo semplicemente sceglie un secondo nodo del cluster e chiede ad esso tutti i contenuti con un trasferimento sul canale privato. Durante questo trasferimento sarebbe logico bloccare il cluster affinchè i contenuti non subiscano alcuna modifica ma questo significherebbe introdurre un'interruzione di servizio! Per questo motivo il trasferimento procede senza imporre lock se non via via sulla singola risorsa trasferita.

Il nodo nuovo entrante deve però nel contempo prestare attenzione ad eventuali comunicazioni di modifica di un contenuto e, nel caso esso fosse stato già precedentemente trasferito, operare come descritto nella sotto-sezione *modify* di *gestione risorse* nella parte successiva.

Come evidente questo approccio prevede che ogni singolo nodo del cluster replichi per intero le risorse esposte dal cluster stesso. In realtà pare più sensato effettuare una replicazione che introduca un minor spreco di spazio disco globale

come ad esempio fa RAID-5 ma tuttavia per semplicità è stata implementata la replicazione RAID-1.

4 Interfacciamento con il Client

4.1 Accesso al Cluster

La fase di accesso al cluster avviene specificando la locazione del medesimo intesa come il canale di comunicazione di gruppo. Come già esposto in 2 questa è l'unica azione non trasparente richiesta al client dopodichè sarà Client Proxy a garantire la connettività in maniera trasparente.

In particolare Client Proxy sceglie arbitrariamente un nodo del gruppo ed effettua la connessione. La possibilità di essere accolto sul nodo sottosta in generale ad una procedura di autenticazione che in questo caso è semplicemente lasciata a sviluppi futuri non tanto perchè complicata come operazione in sè ma perchè si trascina problematiche molto complesse come ad esempio: *dove mantenere la lista degli aventi diritto all'accesso in un sistema come questo?*

4.2 Gestione Risorse

Una volta ottenuto l'accesso al nodo è attraverso esso che il Client Proxy esegue ogni successiva operazione di accesso al file system richiesta dal client.

4.2.1 Listing

Forse la più semplice delle operazioni in un servizio di file system è la richiesta della lista completa delle risorse.

Nel caso, come quello in esame, di replicazione a massima ridondanza (cioè con copie tutte identiche) sarebbe sufficiente che il nodo fornisse direttamente la propria list della directory radice su cui è configurato.

Per rendere il prodotto già *quasi* pronto ad una possibile implementazione del tipo RAID-5 tuttavia il *listing* prevede una interrogazione dell'intero cluster sui contenuti immagazzinati da ogni componente. Nello specifico il nodo su cui si desidera un listing richiede a sua volta a tutti i colleghi la loro lista di risorse e compila, sulla base delle risposte raccolte, la lista completa dei contenuti del cluster. È evidente che in questo modo si ottiene correttamente che, anche in caso di non massima ridondanza, il contenuto del cluster sia uguale per tutti i client indipendentemente dallo specifico nodo cui sono connessi.

Questo scenario risulta quindi molto simile ad un *distributed checkpointing* se non per il fatto che le informazioni derivanti dalla procedura non vengono poi utilizzate per la revisione dello stato interno dei nodi in quanto non significativo.

4.2.2 Read

Per quanto riguarda le successive operazioni bisogna considerare che il sistema deve garantire l'accesso esclusivo alle risorse in caso di richieste concorrenti

di più client. Eccettuato il caso della *Create*, per tutte le altre operazioni il protocollo di mutua esclusione utilizzato è quello di Ricart-Agrawala.

L'accesso in lettura realizzato in questo sistema è un accesso simile a quello delle CVS nel senso che il client acquisisce il diritto di accesso esclusivo al file nel solo periodo di trasferimento del medesimo. Ci si riserva a realizzazioni successive la possibilità di imporre un lock persistente da rilasciare solo su esplicita autorizzazione del client.

4.2.3 Create

Per *Create* vuolsi intendere la modifica del file system attraverso l'introduzione di una risorsa *nuova*, sia essa un file od una directory; si discuterà la scrittura come modifica di un file preesistente nel paragrafo successivo.

Questa operazione a ben vedere non deve precludere altre operazioni di lettura, listing o modifica ma dev'essere comunque eseguita con diritti esclusivi - con lock sulla specifica risorsa da introdurre - al fine di evitare due tipi di inconsistenze: la prima derivante dall'introduzione di una risorsa potenzialmente già esistente nel cluster ma non ancora evidentemente copiata sul nodo; la seconda rappresentata dalla possibilità che due nodi stiano concorrentemente tentando di introdurre la medesima risorsa.

Nello specifico alla richiesta di lock in *create* sulla risorsa un nodo che non presentasse tale risorsa concederà il lock mentre un nodo che dovesse averla notificherà il rifiuto e l'operazione verrà abortita. A ben vedere questo protocollo differisce da quello di Ricart-Agrawala che non prevede veti nell'esecuzione di una procedura. D'altronde se si volesse seguire anche in questo caso tale protocollo si procurerebbe un'attesa infinita sul nodo inziatore.

A seguito dell'introduzione della nuova risorsa è necessario operare la replicazione della stessa. Prima di rilasciare il lock il nodo iniziatore contatta tutti gli altri componenti ed invia il file da aggiungere.

4.2.4 Modify

Diverso è il caso della modifica di una risorsa preesistente. È infatti necessario accordare innanzitutto l'accesso esclusivo a tale risorsa in quanto nel periodo di aggiornamento non deve essere possibile nè leggere la risorsa nè tantomeno effettuare una seconda modifica concorrente.

Ovviamente la modifica non deve avere una dimensione puramente locale, il nodo deve quindi incaricarsi di aggiornare tutte le copie nel cluster prima di rilasciare il lock sulla risorsa medesima.

4.2.5 Delete

La rimozione di un elemento del file system sottosta a tutte le precedenti regole di mutua esclusione in quanto non è naturalmente opportuno rimuovere una risorsa quando un altro processo è in fase di lettura o modifica dalla medesima. Prima di procedere alla rimozione deve essere quindi acquisito il lock su tale risorsa e tutti i nodi che contengono copia di essa devono procedere alla cancellazione.

4.3 Migrazione

L'evento di migrazione in generale avviene per motivi di guasto sul nodo cui il client era connesso e quindi le operazioni sono tutte concentrate sul Client Proxy mentre nulla è in generale lasciato al nodo. Nel caso comunque che la migrazione dovesse essere dovuta ad altri fattori ovviamente anche il nodo che ossera l'abbandono del client deve compiere qualche attività. Ci si può tuttavia ricondurre a quanto espresso nel paragrafo successivo riferito all'abbandono del client.

Per quanto riguarda invece le operazioni da compiere sul Client Proxy, oltre ovviamente alla ricerca di un nuovo nodo ed all'autenticazione su di esso è necessario che tutte le risorse di cui si era riusciti ad acquisire il lock siano considerate come perse. In altri termini ogni lock dev'essere riguadagnato!

4.4 Abbandono del Cluster

Indipendentemente che esso sia *graceful* o meno è importante che il nodo su cui era collegato il client liberi tutti gli eventuali lock acquisiti da esso notificando tali *free* anche ovviamente al resto del gruppo.

5 Conclusioni

In conclusione il sistema realizzato gode di buoni requisiti di affidabilità e disponibilità, tuttavia sia è riscontrata una non elevata scalabilità in quanto a mole dei contenuti immagazzinabili nel file system dovuta essenzialmente alla scarsa funzionalità dell'infrastruttura di rete per il multicast fornita dal sistema operativo.

In particolare si è evidenziato un elevatissimo tasso di perdita di datagrammi nel momento in cui un unica comunicazione dev'essere spezzata su più pacchetti (ad esempio la comunicazione del proprio listing) non permettendo effettivamente il completamento del protocollo.

Sviluppi futuri, oltre che concentrarsi su questo punto fondamentale potrebbero rendere l'interfaccia client più completa ed eventualmente fortemente integrata con il sistema operativo stesso (come succede per NFS).

Indice