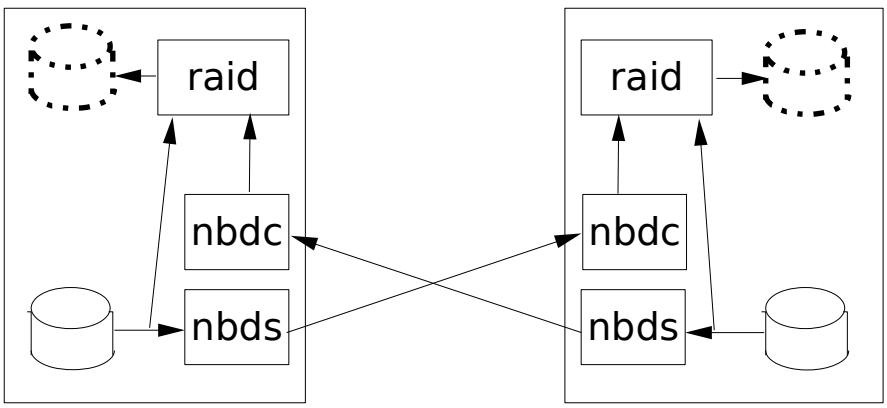
Tecniche per la salvaguardia della disponibilità ed integrità dei sistemi di elaborazione e delle informazioni

2. filesystem distribuiti

Marco Prandini
Università di Bologna

Un approccio olistico

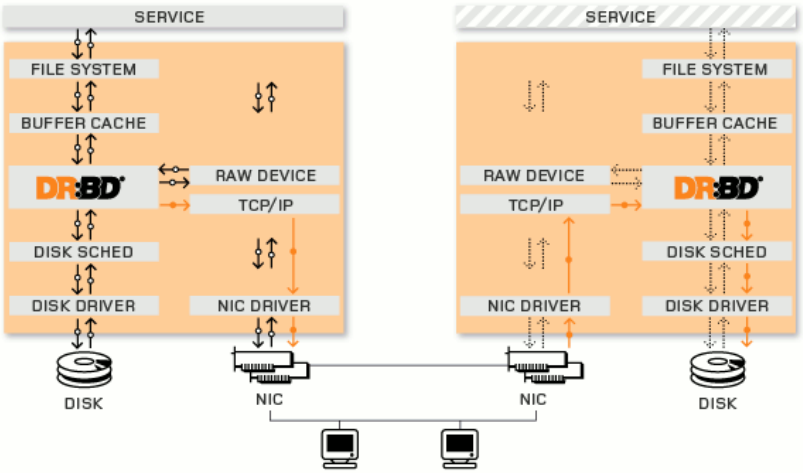
- Anziché ridondare ogni componente (disco, controller, connessione), si può pensare di replicare l'intero sistema di erogazione dei dati
- Modo “artigianale”
  - Il concetto di Virtual File System consente di realizzare strati di astrazione che mostrano qualsiasi sistema di accesso remoto (ssh, ftp, nfs, smb, ...) come un block device
    - prima implementazione: nbd (network block device)
- Protocolli ad-hoc
  - es. DRBD (Distributed Redundant Block Device)



DRBD

- Distributed Redundant Block Device
  - modulo kernel per l'implementazione via VFS
  - strumenti userland per l'amministrazione
- Disponibilità
  - DRBD è prodotto da LINBIT e rilasciato sotto GPLv2
  - È possibile acquistare supporto dal produttore, con diversi livelli di garanzia sui tempi e le tipologie di intervento
- Caratteristiche principali
  - differenti modalità di comunicazione tra host e di scrittura sui dischi
    - compromesso latenza/affidabilità
  - differenti modalità di protezione dell'accesso ai volumi primari / di backup
  - integrazione con sistemi di gestione del cluster

DRBD



DRBD – modi d'operazione

- Primary e Secondary
  - un nodo DRBD è *primary* se può utilizzare il dispositivo virtuale (formato dal mirror dei device locali e dei device dell'altro nodo)
  - un nodo DRBD è *secondary* se, pur partecipando attivamente ad un mirror per replicare i dati tra locale e remoto, non può utilizzare il dispositivo virtuale
- Single-primary mode
  - è l'approccio canonico;
  - ogni risorsa è, in un dato momento, in ruolo *primary* solo su un nodo e *secondary* sull'altro
  - funziona con qualsiasi file system convenzionale (ext3, XFS, reiser etc..).
- Dual-primary mode
  - disponibile in DRBD 8.0 e successive;
  - ogni risorsa è, in un dato momento, in ruolo *primary* su entrambi i nodi del cluster;
  - l'accesso concorrente ai dati deve essere mediato da un lock manager o da un cluster filesystem

DRBD – protocolli di replicazione

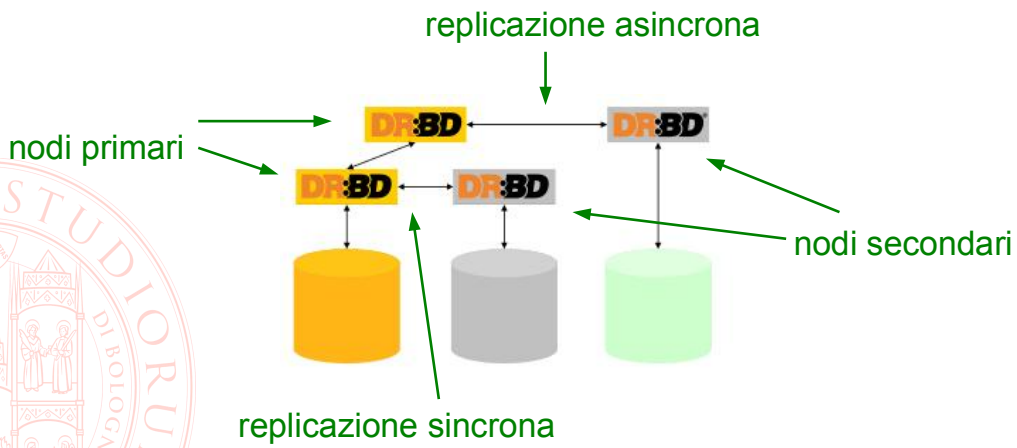
- I diversi protocolli si distinguono per le condizioni richieste per considerare effettivamente riuscite le operazioni di scrittura
  - la scrittura del componente locale del mirror è sempre richiesta
  - i requisiti per considerare completa la scrittura sul componente remoto differenziano i protocolli – maggior garanzia = maggiore latenza!
- Protocollo A (asincrono)
  - ACK se il pacchetto di replicazione è stato messo nel buffer TCP locale
- Protocollo B (semi-sincrono)
  - ACK se il pacchetto di replicazione ha raggiunto il peer node.
- Protocollo C (sincrono).
  - ACK se è stata confermata la scrittura sul disco del peer node

DRBD – (ri)sincronizzazione

- DRBD utilizza metadati per tener traccia dell'allineamento dei blocchi tra i nodi
  - mai utilizzare direttamente i device sottostanti un mirror DRBD!
  - in caso di interruzione del collegamento o di stop di un nodo, i metadati consentono una risincronizzazione molto efficiente
    - nelle ultime versioni (>8.2.0) anche in caso di *split brain*
  - dove supportato, i metadati sono scritti tramite disk flushes

DRBD – resource stacking

- DRBD è considerato uno strumento tipicamente HA
- Per consentire l'implementazione di soluzioni di DR è possibile realizzare un "mirror di mirror"
  - mirror locale tarato sulle alte prestazioni
  - copia remota per resistere ad eventi disastrosi



DRBD – configurazione

- Tutti gli aspetti di DRBD sono gestiti e controllati da un singolo file di configurazione `/etc/drbd.conf`.
- Questo file deve essere uguale su tutti i nodi del cluster.
- Esempio:

sezione common: valori di default

sezioni resource: configurazione delle unità disco

device logico gestito da DRBD

disco locale associato

network port associata

collocazione dei metadati

```
global {
  usage-count yes;
}
common {
  protocol C;
}
resource r0 {
  on alice {
    device    /dev/drbd1;
    disk      /dev/sda7;
    address   10.1.1.31:7789;
    meta-disk internal;
  }
  on bob {
    device    /dev/drbd1;
    disk      /dev/sda7;
    address   10.1.1.32:7789;
    meta-disk internal;
  }
}
```


DRBD – amministrazione

- Il funzionamento delle singole risorse configurate può essere pilotato con il comando

  `drbdadm <azione> <nomerisorsa>`

  - `nomerisorsa` fa riferimento alla stringa dichiarata nell'intestazione resource del file di configurazione
  - può essere "`all`" per applicare l'azione a tutte le risorse configurate
- Azioni fondamentali
  - attivazione e disattivazione
    - `attach`: attiva sul nodo locale la risorsa drbd, collegandola al disco locale – l'operazione opposta viene svolta da `detach`
    - `connect`: attiva la replicazione della risorsa drbd, connettendola al nodo remoto – l'operazione opposta viene svolta da `disconnect`
    - up = attach+connect – down = disconnect+detach
    - `primary`: richiede che il nodo su cui è lanciato il comando diventi primario - l'operazione opposta viene svolta da `secondary`
  - si noti come sia possibile operare sul solo disco locale senza bypassare completamente drbd, ponendolo in modo attached/disconnected

DRBD – amministrazione

- Azioni fondamentali (cont.)
  - monitoraggio
    - role: riporta il ruolo dei nodi (primary o secondary), prima il locale poi il peer
    - cstate: riporta lo stato di connessione della risorsa
    - verify: innesca la verifica on-line della consistenza dei dati
  - riconfigurazione
    - adjust: applica alle risorse i cambiamenti apportati alla configurazione
    - resize: ridimensiona le risorse
    - invalidate(-remote): marca il disco locale (remoto) come out-of-sync

DRBD – il driver

- Il kernel deve essere equipaggiato col modulo drbd
- A modulo caricato, è possibile consultare lo stato dei device configurati con

```
cat /proc/drbd
```

```
version: 8.2.4 (api:88/proto:86-88)
GIT-hash: fc00c6e00a1b6039bfcebe37afa3e7e28dbd92fa build by
buildsystem@linbit, 2008-01-11 12:44:36
0: cs:Connected st:Secondary/Secondary ds:UpToDate/UpToDate C r---
   ns:524288 nr:524288 dw:524288 dr:524288 al:0 bm:64 lo:0 pe:0 ua:0 ap:0
  resync: used:0/31 hits:524224 misses:64 starving:0 dirty:0 changed:64
  act_log:used:0/257 hits:0 misses:0 starving:0 dirty:0 changed:0
```

stato del disco locale/remoto

stato del nodo locale/remoto

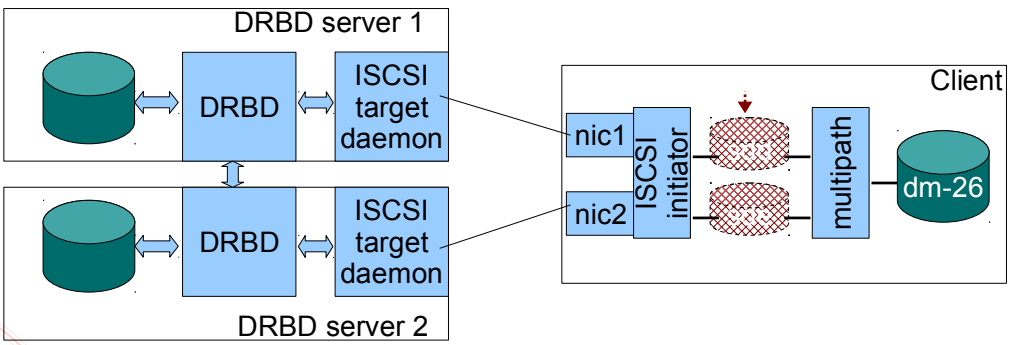
stato della connessione tra i nodi

DRBD – un esempio di upgrade HW dello storage con downtime limitato

- Supponiamo di avere due file server A (attivo) e B (backup) e di voler sostituire i dischi ormai pieni con altri più capienti
- Possibile sequenza:
  1) check that all the services are on server A
  2) replace the disk on server B
  3) restart drbd on server B, with the bigger local devices
     - create new metadata may be needed
  4) wait for A->B resync end
  5) transfer all the services on server B
  6) replace the disk on server A
  7) restart drbd on server A, with the bigger local devices
     - create new metadata may be needed
  8) wait for B->A resync end
  9) live grow the filesystem on server B with resize2fs
     - if done in disconnected mode, will keep a copy of the original fs on server A
     - new B's fs can be tested and, if ok, synced to A at reconnect

Un esempio di cose da NON fare

- Si potrebbe essere tentati di combinare le tecniche viste per realizzare una SAN HA iSCSI
  - sfruttando la configurazione dual primary per avere la possibilità materiale di esporre contemporaneamente il volume DRBD dai due nodi
  - utilizzando il multipath in modo active/standby per evitare scritture concorrenti



- Bisogna però stare sempre attentissimi a come i vari layer usano caching, stateful communication e metadati!

http://fghaas.wordpress.com/2011/11/29/dual-primary-drbd-iscsi-and-multipath-dont-do-that/

  - La soluzione corretta è avere UNA sola "testa" (server) mantenuta in vita in caso di guasto

NFS

- La prima implementazione di NFS (Sun Microsystems) risale al 1985 .
  - La versione 2 è stata quella effettivamente utilizzata per più tempo.
  - La versione 3 ha risolto in gran parte i gravi problemi di prestazioni.
  - La versione 4 ha visto l'introduzione di numerose caratteristiche innovative sui fronti:
    - sicurezza e controllo dell'accesso
    - replicazione e migrazione
    - utilizzo in reti topologicamente complesse

NFS – Approccio originale

- Server senza stato (ogni operazione richiesta dal client è autocontenuta)
- Componenti separati per i diversi servizi
  - mount
  - condivisione
  - locking
  - quota enforcement

NFS – Protocolli

- Tutte le operazioni di NFS sono implementate come chiamate a procedura remota (RPC).
- Questa scelta consente di espandere con grande flessibilità il set di demoni incaricati delle varie funzioni, e di nascondere i dettagli del protocollo di trasporto sottostante (originariamente UDP, ora comunemente TCP).
  - Il demone `portmap` gira su una porta fissa (111 tcp ed udp)
  - I demoni `nfsd`, `mountd`, `lockd`, `statd`, `rquotad` hanno un numero di servizio RPC prefissato
  - All'avvio teoricamente potrebbero cercare una qualsiasi porta libera e registrarsi presso portmap

NFS – Il lato server

- I demoni essenziali sono `mountd` per effettuare le operazioni di mount e `nfsd` per la gestione dei file.
  - Il compito del primo è rilasciare semplicemente un cookie al client, col quale questo può indicare ad nfsd su quale filesystem vuole eseguire una data operazione.
  - I mount attivi sono memorizzati in `/var/lib/nfs/rmtab`
- L'elenco delle directory esportate è in formato testo in `/etc/exports`
  - Ad ogni modifica si deve lanciare il comando exportfs che lo converte nel database per il controllo degli accessi binario `/var/lib/nfs/xtab`

NFS – /etc/exports

```
/cs/users           host1(ro) host2(rw)
/usr/share/man      *.unibo.it (ro)
```

Client che possono accedere — Opzioni di accesso

- Specifiche valide per identificare i client:
  - hostname
  - @netgroup
  - wild card
  - IP nets (CIDR)
- Lato client, si può verificare se esiste un export con

```
showmount -e nomeserver
```


NFS – sicurezza

- Non c'è alcun meccanismo di autenticazione degli host (solo ip- e name-based, eventualmente un po' di aiuto da *tcpd*)
- Meccanismi molto rudimentali di contenimento degli utenti
  - Gli UID e GID sono globali (cosa succede se l'utente **dante** su di un client monta una dir sul server dove lo stesso UID è dell'utente **virgilio**? → necessità, nelle grandi reti, di far coesistere NFS con sistemi di distribuzione degli account)
  - Vale anche per root!
    - Sia nel senso che root su di un client può impersonare qualsiasi UID
    - Sia nel senso che root su di un client potrebbe assumere i diritti di root su file del server

NFS – Opzioni di condivisione

- Alcune opzioni di condivisione possono mitigare i problemi illustrati ed agire sul tuning delle prestazioni. Tra queste:

Diritti per host:	`ro, rw, rw=list`
Mappatura utenti:	`root_squash, no_root_squash, all_squash, anonuid=xxx, anongid=xxx`
Verifica privilegi client:	`secure, insecure`
Prestazioni:	`sync, async, wdelay, no_wdelay`
Restrizioni sulle subdir:	`noaccess`

NFS – Prestazioni, tuning e verifica

- Una volta effettuato il mount, la gestione dei file viene condotta da nfsd. Un elemento cruciale è il numero di fork di nfsd:
  - non devono essere molti perché vengono risvegliati tutti dal kernel all'arrivo di una richiesta
  - non devono essere pochi per evitare problemi di overflow dei socket UDP

    come verificare: `ss -s` , `nfsstat [-c|-s]`

NFS – Lato client

- Il mount viene effettuato come per un file system locale, usando

```
mount -t nfs myserver:/exported/path /myserver/path
```

  e naturalmente può essere automatizzato inserendo gli stessi parametri in `/etc/fstab`

- Convenzione consigliata: comporre il path del mount point in modo che comprenda il nome del server (come nell'esempio sopra)

NFS – Lato client

- Opzioni di mount:
  - dimensione dei blocchi trasferiti: `rsize, wsize`
  - porta se non standard: `port`
  - permessi: `ro, rw`
  - protocollo: `timeo, retrans, retry`
  - esecuzione del mount: `bg, fg`
  - esecuzione delle operazioni: `hard, soft, (no)intr`

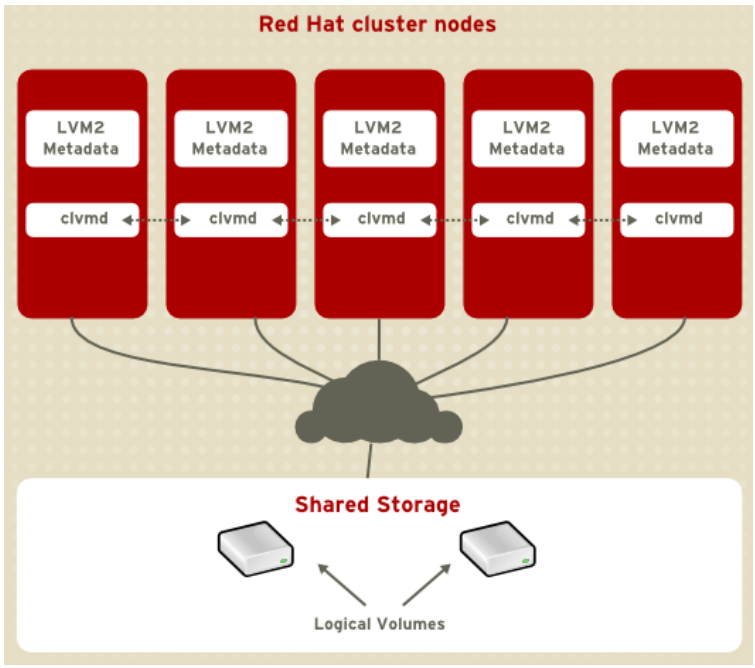
Cluster Filesystems

- Se si vuole accedere direttamente ad un block device (SAN) da molti client, serve un meccanismo di arbitraggio distribuito: un *cluster filesystem*
- Un cfs si occupa contemporaneamente
  - di organizzare i file sul block device, come un normale fs
  - di gestire l'accesso concorrente dai diversi client

  ed eventualmente
  - di replicare I dati su nodi diversi
    - per garantire tolleranza ai guasti (es. Andrew)
    - per consentire l'accesso anche a nodo disconnesso dallo storage principale (CODA)
    - per fornire migliori prestazioni tramite l'accesso parallelo (es. GPFS)
- Vediamo come esempio i diversi componenti di RedHat GFS

Cluster Logical Volume Manager

- L'utilizzo di un dispositivo fisico accessibile via rete (SAN-style) può godere delle caratteristiche di flessibilità garantite da LVM purchè I metadati siano resi noti a tutti I nodi
- → CLVM
  - LVM per l'organizzazione dello spazio sul disco
  - *demone clvmd* per la distribuzione dei metadati LVM ai client
- CLVM non fa locking!



Distributed Lock Manager

- DLM è il sistema di locking alla base di GFS
- Componenti
  - modulo del kernel per l'integrazione del locking nel VFS
  - *dlm_controld*: demone per l'interazione con lo userspace ed il sistema di gestione del cluster
  - *libdlm*: libreria per la costruzione di applicazioni dlm-aware
- Funzionamento essenziale
  - permette il locking di qualsiasi risorsa (es. un file, un intero DB, un record di un DB, una struttura dati,...)
  - *advisory lock*, non previene forzatamente l'accesso alle risorse ma permette alle applicazioni di coordinarsi
  - Il nodo che per primo cerca di accedere ad una risorsa ne diventa *lock master*: conserva la copia principale delle informazioni di locking, che sono poi distribuite anche agli altri nodi
  - le richieste di locking sono sempre servite attraverso la consultazione del lock master, ma le copie consentono di operare correttamente se un nodo cade

Distributed Lock Manager

- Le tipologie di lock implementate da DLM sono le seguenti:
  - Null lock (NL)
  - Concurrent Read (CR)
  - Concurrent Write (CW)
  - Protected Read (PR)
  - Protected Write (PW)
  - Exclusive (EX)

	NL	CR	CW	PR	PW	EX
NL	SI	SI	SI	SI	SI	SI
CR	SI	SI	SI	SI	SI	NO
CW	SI	SI	SI	NO	NO	NO
PR	SI	SI	NO	SI	NO	NO
PW	SI	SI	NO	NO	NO	NO
EX	SI	NO	NO	NO	NO	NO

- La tabella riportata indica se, in presenza del lock indicato nella riga, può essere concesso ad un'altra applicazione il lock indicato nella colonna

Global File System

- GFS è il cfs proposto da RedHat
- Componenti
  - modulo kernel per il VFS
  - gfs_controld: demone per l'interazione con lo userspace ed il sistema di gestione del cluster
- La struttura su disco è simile a quella di ext3
  - superblock
  - resource groups (invece dei block groups)
    - in ogni rg
      - copia ridondata del superblock
      - una parte della tabella degli inode
      - data blocks
    - ridondanza + maggior località
  - journal: almeno tanti quanti i nodi

Global File System

- Per effettuare il locking, GFS si appoggia
  - su un lockmanager fittizio se c'è un solo nodo
  - su DLM altrimenti
- I lock possono essere di tre tipi, su vari tipi di risorsa
  - exclusive → dlm_EX
  - shared → dlm_PR
  - deferred → dlm_CW
  - nodo
  - journal
  - metadati
  - file
  - …
- GFS funziona solo su RedHat Cluster
  - all'atto della creazione si definisce una *locktable*
    - contiene il nome del cluster i cui nodi potranno effettuare il mount