

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

3. clustering

Marco Prandini
Università di Bologna



Disponibilità continua dei sistemi

- L'affidabilità dello storage è solo uno dei componenti per la disponibilità dell'intero sistema
 - alimentazione
 - memoria
 - chipset e processore
 - ventilazione
 - schede
 - connessione alla rete
- Ognuno di questi componenti può essere ridondato
 - l'architettura risultante può essere di straordinaria complessità e costo
 - il vantaggio è quello di avere a che fare con un unico sistema
 - amministrazione semplificata
 - nessuna problematica di coerenza tra dati replicati



Disponibilità continua dei sistemi

■ Approcci alternativi nella definizione delle risorse logiche:

- *virtualizzare* il sistema per poterlo riprodurre su di un host diverso semplicemente copiando una manciata di file
 - molteplicità di host, ma con poche problematiche di coerenza
 - beneficio collaterale: server consolidation
 - ottimo isolamento tra le diverse VM
 - buona scalabilità, anche se l'overhead per la virtualizzazione di servizi semplici è elevato
- realizzare più copie dello stesso sistema, ciascuna con tecnologie standard: *cluster HA* (high availability)
 - economico ed efficiente
 - più problematica la scalabilità

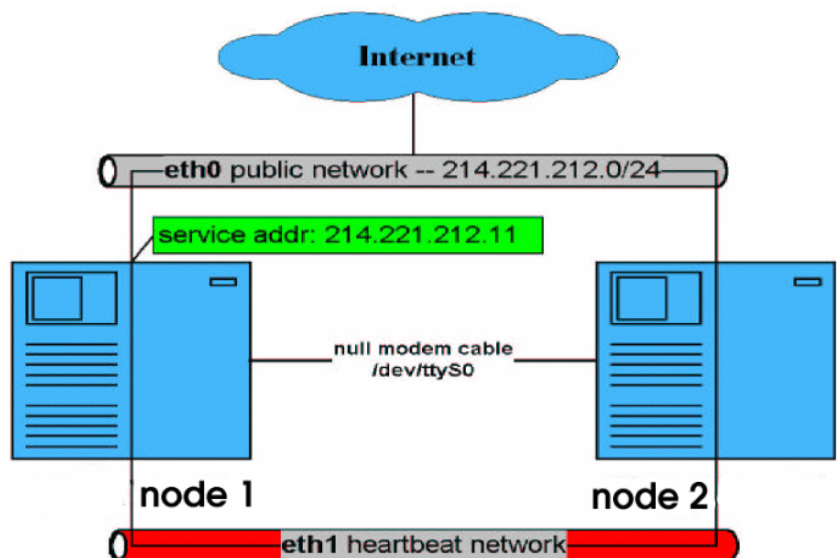
■ In ogni caso serve una tecnologia per gestire automaticamente lo spostamento delle risorse logiche sui nodi fisicamente disponibili

3

Heartbeat

■ Segnalazione della vitalità

- Il modo più semplice di rilevare la necessità di sostituire una risorsa guasta è scambiare periodicamente pacchetti di vitalità (*heartbeat*)
 - è sicuramente consigliabile farlo attraverso link multipli per non confondere il fallimento del link con quello del nodo
 - non è sufficiente per capire se il servizio viene erogato correttamente
 - monitor aggiuntivi per la verifica di risposte sensate da parte dei server applicativi



4

Failover/Failback

- La scomparsa e ricomparsa di nodi attivi richiede la gestione di due condizioni:
- **Failover**: la situazione in cui un nodo si guasta e le risorse del cluster lo devono sostituire. Problemi:
 - decidere correttamente chi è guasto, gestendo le situazioni di incomunicabilità (**split brain**)
 - evitare il conflitto di possesso delle risorse
- **Failback**: la situazione in cui un nodo precedentemente guastatosi ripristina la sua funzionalità. Problemi:
 - mantenere le sessioni in corso e ridurre i tempi di commutazione superflui...
 - “nice failback”, chi ha le risorse eroga il servizio
 - ... o sfruttare al meglio la capacità computazionale derivante dai nodi ritornati disponibili
 - “auto failback”, chi è dichiarato “master” per una risorsa la riprende appena torna online

5

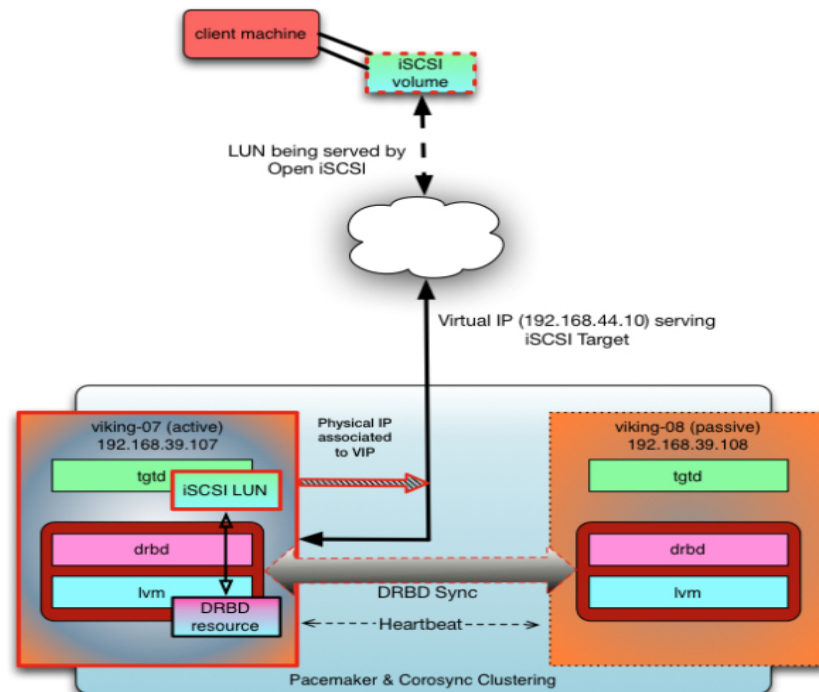
Un esempio di failover: IP address takeover

- La prima fase successiva alla rilevazione del guasto di un nodo è tipicamente la presa in carico, da parte di un altro, della sua identità di rete
- Questo può essere fatto a 3 livelli diversi di rete – in ordine dal più rapido ma complesso al più lento ma semplice:
 - **MAC takeover** (con schede e SO predisposti... ma gli switch?)
 - non sempre applicabile
 - **DNS reconfiguration** (...resolver cache)
 - latenza non affidabilmente controllabile
 - **IP takeover** (...arp cache)
 - la scelta più comune
 - Per non pregiudicare la raggiungibilità dei nodi e per consentire di distinguerli senza ambiguità, ad ogni nodo viene assegnato un IP address reale (fisso), ed al cluster un IP “collettivo” (**floating/virtual/cluster IP**), che viene assunto come alias dal nodo master attivo.

6

Un esempio di failover: HA iSCSI

- Una volta configurato un floating IP, il nodo che lo detiene eroga attraverso questo i servizi, ad esempio un target iSCSI
- Uso corretto di DRBD+iSCSI: il target daemon reinizializza la propria state machine al failover



<http://blogs.mindspew-age.com/2012/04/05/adventures-in-high-availability-ha-iscsi-with-drbd-iscsi-and-pacemaker/>

7

Partizionamento

- Situazione ideale di un cluster
 - ogni nodo è in una di due condizioni:
 - disconnesso dagli altri e senza possibilità di accedere a risorse condivise
 - perfettamente funzionante
- Situazione reale
 - un guasto può *partizionare* l'insieme dei nodi, creando sub-cluster
 - in un sub-cluster i nodi si vedono perfettamente
 - non c'è comunicazione tra un sub-cluster e l'altro
 - ogni sub-cluster mantiene un certo grado di operatività
 - può accedere a risorse condivise
 - deve decidere se farlo o meno!
 - anche in assenza di guasti i transitori rendono possibile determinare *solo con un certo grado di probabilità*, da un nodo:
 - se questo sta tentando di associarsi al sub-cluster più opportuno
 - se questo dispone di una visione del sub-cluster omogenea a quella degli altri nodi

8

Fencing

- Un modo per garantire che un sub-cluster non acceda alle risorse condivise è “recintarlo”
 - resource fencing: ogni nodo può privare gli altri dell'accesso ad una precisa risorsa
 - e.g. spegnere la porta dello switch con cui il nodo accede ad una SAN per proteggere un disco condiviso
 - node fencing
 - e.g. spegnere il nodo tramite una scheda di gestione remota (Dell DRAC, SUN/HP iLOM), l'UPS o la power strip che alimenta il sistema



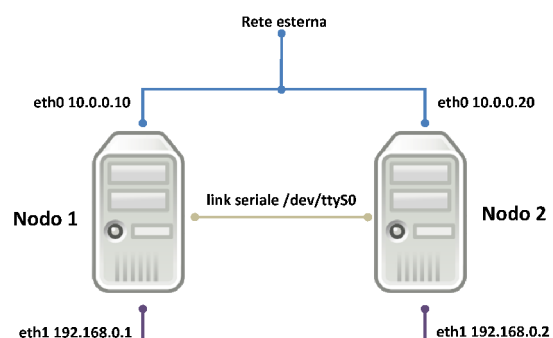
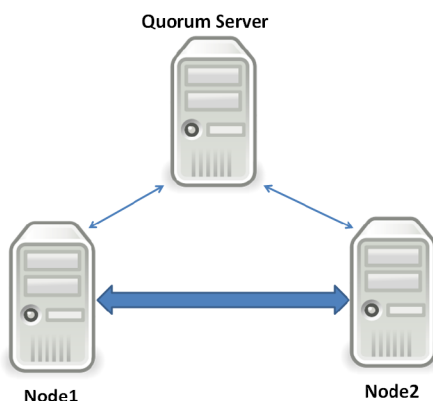
- STONITH (Shoot The Other Node In The Head)

- Chi decide in caso di autentico partizionamento (non di guasto del nodo) chi deve effettuare il fencing e chi lo deve subire?

9

Quorum

- Nei cluster con $N > 2$ membri, è possibile adottare un criterio semplice e robusto per determinare quale sub-cluster debba ritenersi attivo
 - ogni nodo ha un voto
 - se un sub-cluster raggiunge $V > N/2$ voti, raggiunge il quorum ed ogni nodo che ne fa parte può operare
 - non può esistere più di un sub-cluster che detiene il quorum
- Nei cluster a due nodi, questo sistema previene il funzionamento del nodo superstite quando l'altro fallisce. Si può operare
 - o tramite sistemi di arbitraggio esterni al cluster (fisicamente o logicamente)
 - o contando su ridondanze hardware che garantiscano la comunicazione tra i nodi quando questi sono operativi, per poi consentire il funzionamento con $V=1$



10

Sistemi di clustering HA per Linux

■ Linux-HA (<http://www.linux-ha.org/>)

- a volte chiamato Heartbeat per il suo componente più anziano e più evidente, è il sistema di HA-clustering open source più stabile (1999)
- per pulizia ed interoperabilità, il sottosistema di cluster resource management è stato separato, dando origine al progetto Pacemaker (<http://clusterlabs.org/wiki/Pacemaker>)

■ OpenAIS (<http://www.openais.org/>)

- implementazione delle specifiche pubbliche AIS (Application Interface Specification) del Service Availability Forum
 - interfacce e politiche standard per lo sviluppo di applicazioni HA
 - <http://www.saforum.org/> → specifications → AIS
 - pensato per cooperare con Corosync cluster engine (<http://www.corosync.org/>)

■ Sistemi sviluppati da vendor

- RedHat Cluster Suite, Novell/SuSE Cluster Suite, Symantec Veritas Cluster Server, IBM Cluster Systems Management, HP Serviceguard, ...

■ <http://www.squidoo.com/linux-clustering>

11

Management per i sistemi open – DMC

The screenshot displays the DRBD Management Console (DMC) interface for a cluster named 'UbuntuClusterHA'. The interface is divided into three main sections:

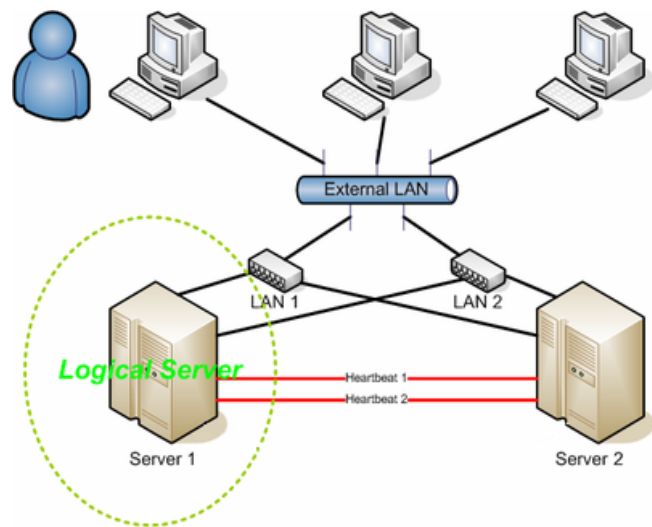
- Left Sidebar:** A tree view showing the cluster hierarchy. The 'Cluster' section is expanded, showing 'Cluster Wizard' and 'Advanced Mode'. Below this, the 'Storage (DRBD)' section is expanded, showing 'drbd: nfsexport' and 'drbd: /dev/sdb2'. The 'Cluster' section is also expanded, showing 'Available Services' and 'Services'. The 'Services' section is expanded, showing 'IPAddr2 (1 / 192.168.56.100)', 'Filesystem (/dev/drbd0)', 'Master/Slave Set (drbd_1)', 'Group (1)', 'nfs-kernel-server (1)', and 'SendArp (1)'.
- Central Diagram:** A diagram showing the cluster topology. It includes two nodes, 'ubuntu2' and 'ubuntu1', both marked as 'online'. A 'linbit: drbd (nfsexport)' service is shown with 'Master/Slave Set (drbd_1)' and 'primary on: ubuntu2' and 'secondary on: ubuntu1'. A 'Filesystem (/dev/drbd0)' service is shown with 'running on: ubuntu1' and '(migrated)'. A 'Group (1)' service is shown with 'running on: ubuntu1' and 'SendArp (1)' and 'nfs-kernel-server (1)'. Arrows indicate connections between the nodes and the services.
- Right Panel:** A configuration panel for the selected service. It includes tabs for 'Primitive', 'Clone', and 'Master/SL...'. The 'Primitive' tab is active, showing fields for 'ID' (1), 'Heartbeat' (grp_1), 'Provider' (group), 'Class' (group), and 'Host Locations' (on ubuntu2 and on ubuntu1).

At the bottom of the interface, there is a terminal window showing the command line interface for the cluster configuration. The commands shown are:

```
node=ubuntu2 score=0 />
[root@ubuntu2:~#] /usr/sbin/cibadmin --obj_type constraints -C -X '<rsc_location id="loc_res_SendArp_1_ubuntu1" rsc="res_SendArp_1"
node="ubuntu1" score="0"/>'
[root@ubuntu2:~#] /usr/sbin/cibadmin --obj_type constraints -C -X '<rsc_colocation id="col_res_Filesystem_2_grp_1" score="INFINITY" rsc="grp_1"
with-rsc="res_Filesystem_2"/>'
[root@ubuntu2:~#] /usr/sbin/cibadmin --obj_type constraints -C -X '<rsc_order id="ord_res_Filesystem_2_grp_1" score="INFINITY"
first="res_Filesystem_2" then="grp_1"/>'
```

Linux-HA

- La soluzione più tradizionale per i cluster a 2 nodi Active/Standby
- Nella versione 2 supporta anche più di 2 nodi



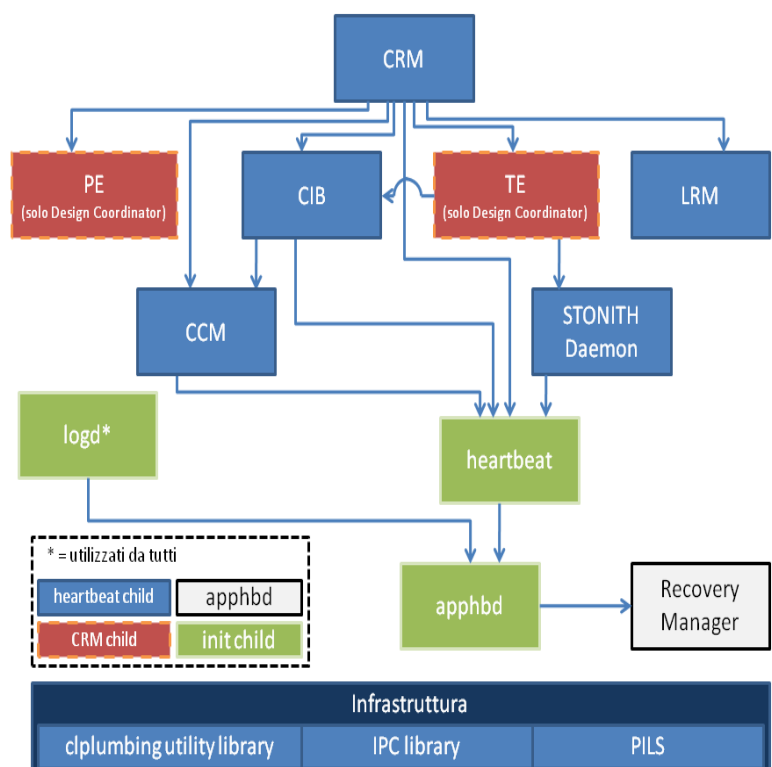
- Eccellente tutorial:

http://www.linux-ha.org/_cache/HeartbeatTutorials__LWCE08-ha-tutorial.odp
da <http://www.linux-ha.org/HeartbeatTutorials>

13

Linux-HA: architettura

- Heartbeat: comunicazione tra i nodi
 - CRM: Cluster Resource Manager
 - PE: CRM Policy Engine
 - TE: CRM Transition Engine
 - CIB: Cluster Information Base
- CCM: Strongly Connected Consensus Cluster Membership
- LRM: Local Resource Manager
- Stonith Daemon: fencing
- Logd: demone non bloccante per i log
- Apphdb: servizio di watchdog timer a livello applicazione
- Recovery Manager: servizio di recovery delle applicazioni
- Infrastruttura
 - PILS: Plugin and Interface Loading System
 - IPC: Interprocess Communication
 - Cluster “plumbing” library
- CTS: Cluster Testing System, suite di test per stressare il cluster



14

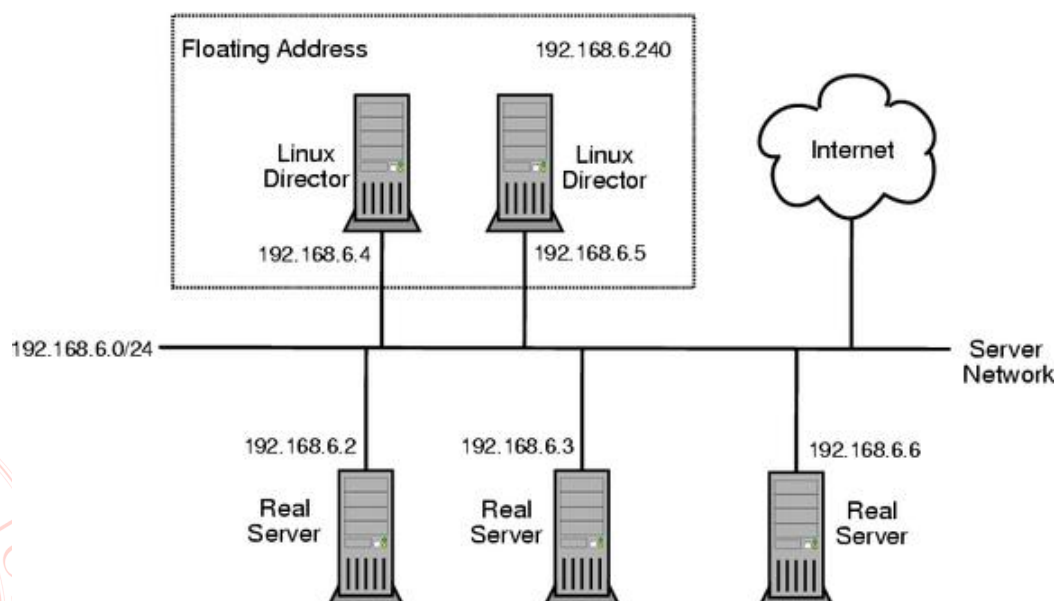
HA + LB

- Con Linux-HA non è facilissimo realizzare strutture scalabili che bilancino il carico su di un pool espandibile di nodi
- Il componente ideale per questo è **LinuxVirtualServer (LVS)**
 - *director* per smistare le richieste
 - si presenta all'esterno come unico *virtual server*
 - ridirige le richieste secondo un algoritmo di bilanciamento ai *real server*
 - diverse strategie di instradamento delle risposte
 - NAT
 - Tunnelling
 - Direct Routing
- I real server non hanno esigenze di HA, poichè sono tutti intercambiabili per costruzione
- Il director invece è un SPOF, ma non ha esigenze spinte di scalabilità

15

HA + LB

- Soluzione: ridonarlo in modo semplice con Linux-HA



16

Red Hat Cluster Suite

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Cluster_Suite_Overview/

- RedHat, la prima azienda nella storia di Linux ad offrire supporto commerciale ad una distribuzione, produce una delle piattaforme server più diffuse: Red Hat Enterprise Linux
 - esiste anche per desktop
 - RH ha cessato il supporto alla distribuzione open, che ha forkato Fedora Core
 - RH partecipa attivamente alla comunità di FC
 - FC è il testbed per molte novità di RHEL
 - CentOS, prendendo dal software senza problemi di licenza di FC, distribuisce gratuitamente un “clone open” di RHEL

■ RHCS fornisce con pochi componenti nativi integrati da progetti open source

sistema di clustering
sistema di load balancing
sistemi di gestione
sistemi di monitoraggio
cluster filesystem

ccs, cman, fence, rgmanager
lvs (pulse, nanny)
system-config-cluster, conga (luci, ricci)
zabbix
clvm, gncbd, dlm, gfs

17

Red Hat Cluster Suite - componenti

- Cluster Configuration System (CCS)
 - mantiene le direttive di configurazione sincronizzate tra i nodi
 - Tutte in un file XML `/etc/cluster/cluster.conf`
- Cluster MANager (CMAN)
 - gestisce join, leave e segnalazioni di vitalità dei nodi
 - calcola il quorum e quindi se un nodo può erogare servizi
- Fence Daemon
- Resource Group Manager (Rgmanager)
 - definisce i subset di nodi su cui può essere collocato un servizio (all'avvio, al failover, al failback)
 - permette di gestire manualmente i servizi

18

Sintetizzando...

■ I sistemi di clustering

- hanno come obiettivo l'erogazione in alta disponibilità delle risorse
- adottano un approccio **totalmente decentralizzato**
 - nella determinazione dei nodi disponibili
 - nella collocazione dei servizi sui nodi

■ L'approccio decentralizzato **non scala** bene come si pensava

- alto overhead ed alta sensibilità dei sistemi di heartbeat
- evoluzione verso **sistemi cloud**
 - **sistemi chiave** (es. gestione risorse, image repository)
non hanno problemi di carico, solo di HA
→ **replicati in modalità cluster minimale (2 nodi)**
 - numero arbitrario di **compute nodes** su cui allocare i servizi

■ In ogni caso...

- in caso di guasto (recuperabile), il downtime tipico è **≥ 1 minuto**
- **il tempo necessario a dichiarare *failed* un nodo non può essere troppo basso, o errori transitori innescerebbero *fencing wars* letali**
- la risorsa, tipicamente una VM, ha tempi di avvio propri (boot + eventuale riparazione del filesystem)

19

Approcci alternativi – hardware

■ Con le architetture cluster non possiamo garantire i five nines.

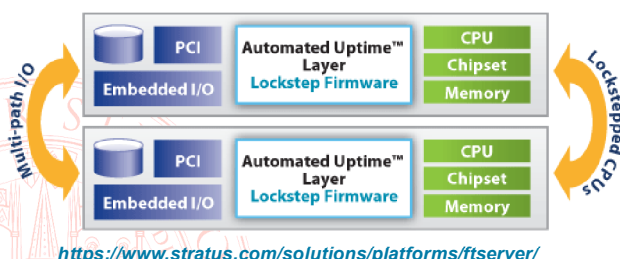
- ma il loro rapporto prezzo/prestazioni è eccellente per la maggior parte delle applicazioni (ad esempio, con la migrazione live delle VM, i downtime pianificati possono durare meno di 1 secondo).

■ Se veramente serve di più, bisogna ricorrere ad approcci hardware

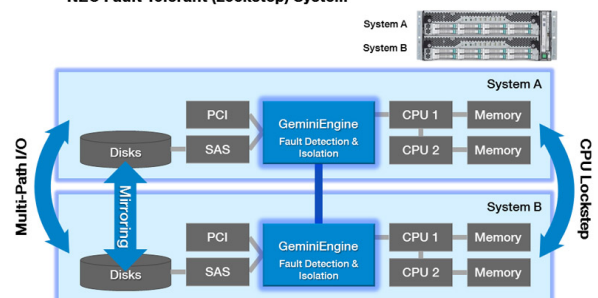
- mirroring dell'intera architettura di calcolo basata su chipset proprietari che riproducono tutte le operazioni identicamente su processori diversi

■ Ovviamente molto costosi, e poco scalabili

Duplex Hardware Components

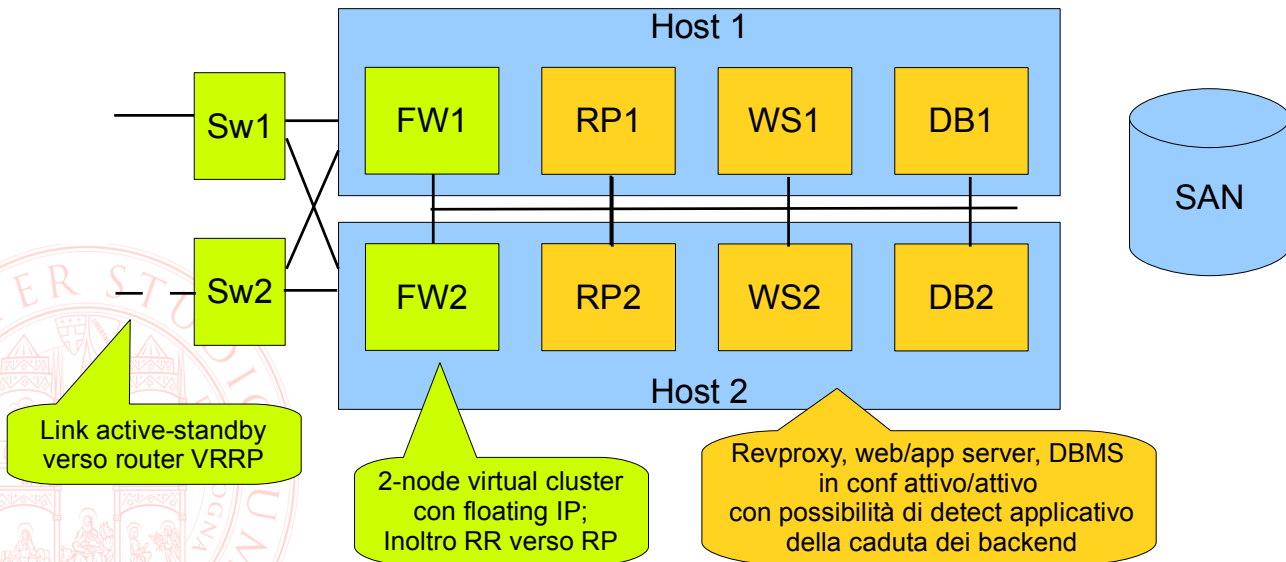


NEC Fault Tolerant (Lockstep) System



Approcci alternativi – software

- Dove è possibile, si può ricorrere alla ridondanza per eliminare completamente i single point of failure a livello applicativo
 - La virtualizzazione aiuta molto
 - Serve grande cura nella scelta delle configurazioni e nel tuning
 - Non tutto è replicabile senza problemi e con le stesse prestazioni



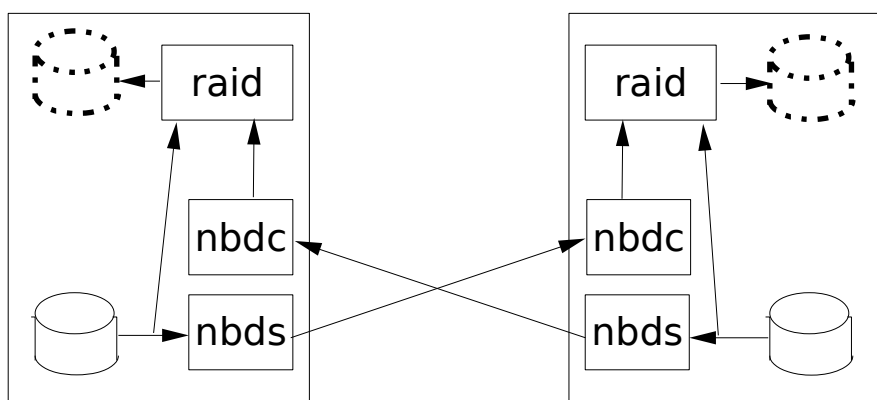
Tecniche per la salvaguardia della disponibilità ed integrità dei sistemi di elaborazione e delle informazioni

4. 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

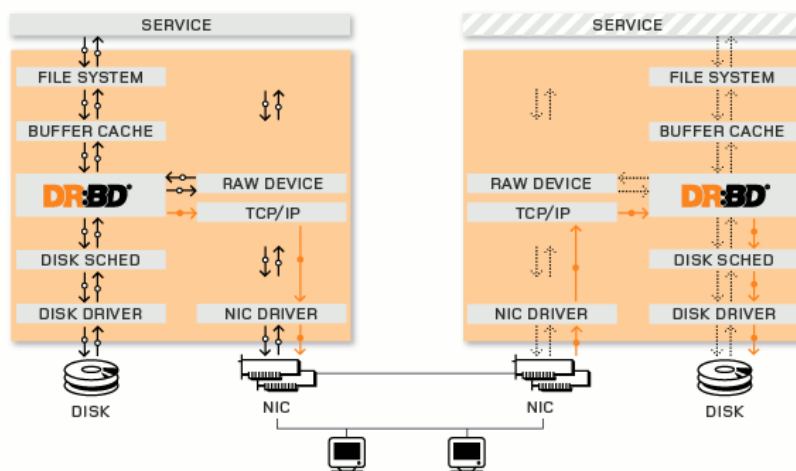
- DRBD è prodotto da LINBIT e rilasciato sotto GPLv2
- È possibile acquistare supporto dal produttore, con diversi livelli di garanzia sui tempi e le tipologie di intervento

■ 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

3

DRBD



4

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

5

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

6

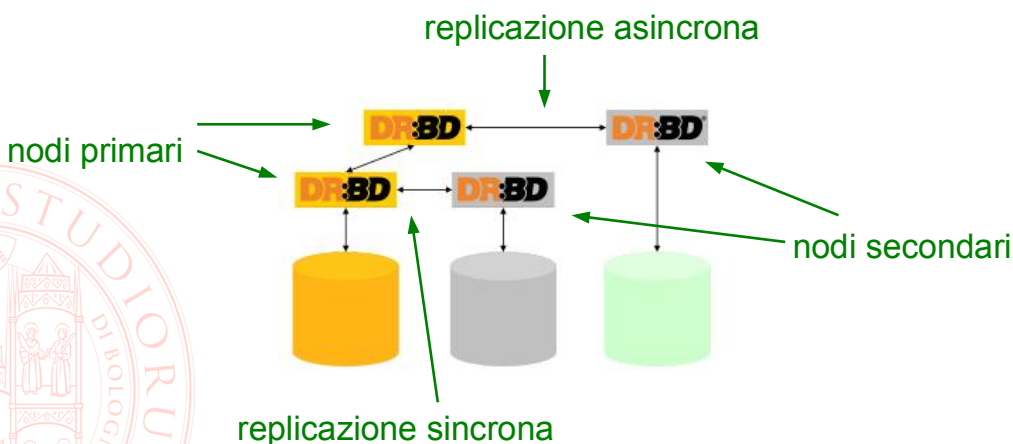
DRBD – (ri)sincronizzazione

- 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

7

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



8

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;
    }
}
```

9

DRBD – amministrazione

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

`drbdadm <azione> <numerisorsa>`

- `numerisorsa` 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`

10

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

11

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

12

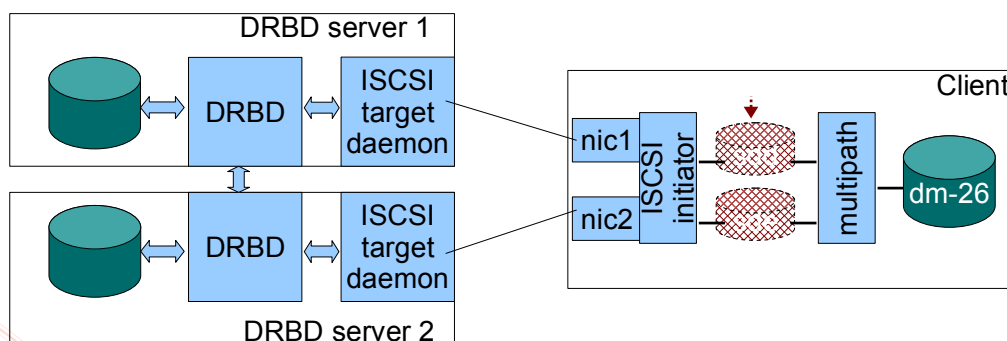
DRBD – un esempio di upgrade HW dello storage con downtime limitato

- 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

13

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

14

■ 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

17

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**

18

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

19

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

20

NFS – Opzioni di condivisione

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

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

21

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]`

22

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)

23

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

24

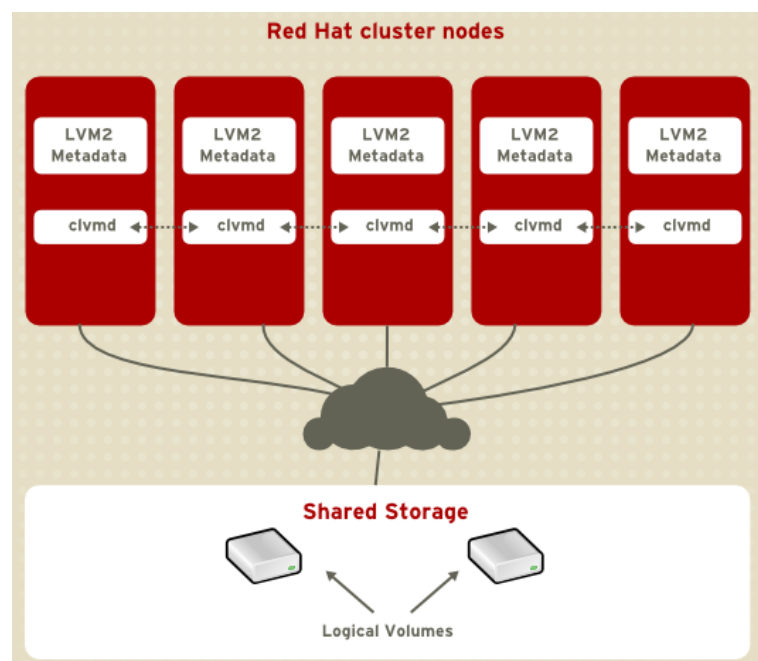
Cluster Filesystems

- 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 cliented 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

25

Cluster Logical Volume Manager

- L'utilizzo di un dispositivo fisico accessibile via rete (SAN-style) può godere delle caratteristiche di flessibilità garantite da LVM purché 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!



26

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_control*: 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

27

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

28

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

29

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

30