High Availability in Linux Systems

Marco Stornelli

Rights to copy

creative commons
COMMONS DEED

Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

Attribution. You must give the original author credit.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: http://creativecommons.org/licenses/by-sa/3.0/legalcode

Marco Stornelli

http://www.webalice.it/firefox82

Best viewed with...

This document is best viewed with a recent PDF reader or with OpenOffice.org itself!

- Take advantage of internal or external hyper links
  So, don't hesitate to click on them! See next page.
- Find pages quickly thanks to automatic search
- Use thumbnails to navigate in the document in a quick way

Course prerequisites

Skills to make these lectures profitable

Familiarity with Unix concepts and its command line interface

- Essential to manipulate sources and files
- Essential to understand and manage the system that you build

Contents

High Availability overview

High Availability concepts

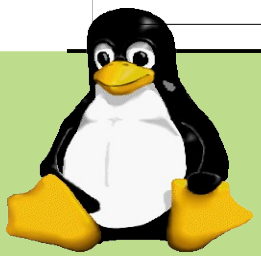
High Availability clustering

- Putting together a group of computers which trust each other to provide a service even when system components fail
- When one machine goes down, others take over its services/resources
- Resources can be IP addresses, disk partitions, etc.
- Services can be web servers, DB servers, etc.

NINEs

- Availability in terms of numbers of nine:

Availability Class	Availability	Downtime per year	Type
1	90%	36,5 days	Unmanaged
2	99%	3,65 days	Managed
3	99,9%	9 h	Well-managed
4	99,99%	52 min	Fault-tolerant
5	99,999%	5 min	Highly available
6	99,9999%	30 sec	Very highly available
7	99,99999%	3 sec	Ultra available



High Availability limits

- Can I have a system availability of 100%? **NO**, you can't
- Number of faults: usually HA systems designed to manges single faults
- Recovery speed: usually from about a second to a few minutes
- A good HA clustering system adds a “9” to your base availability
  - 99->99.9, 99.9->99.99, 99.99->99.999, etc.

High Availability issues

- Hardware costs
- Software costs
- Complexity
- Standards

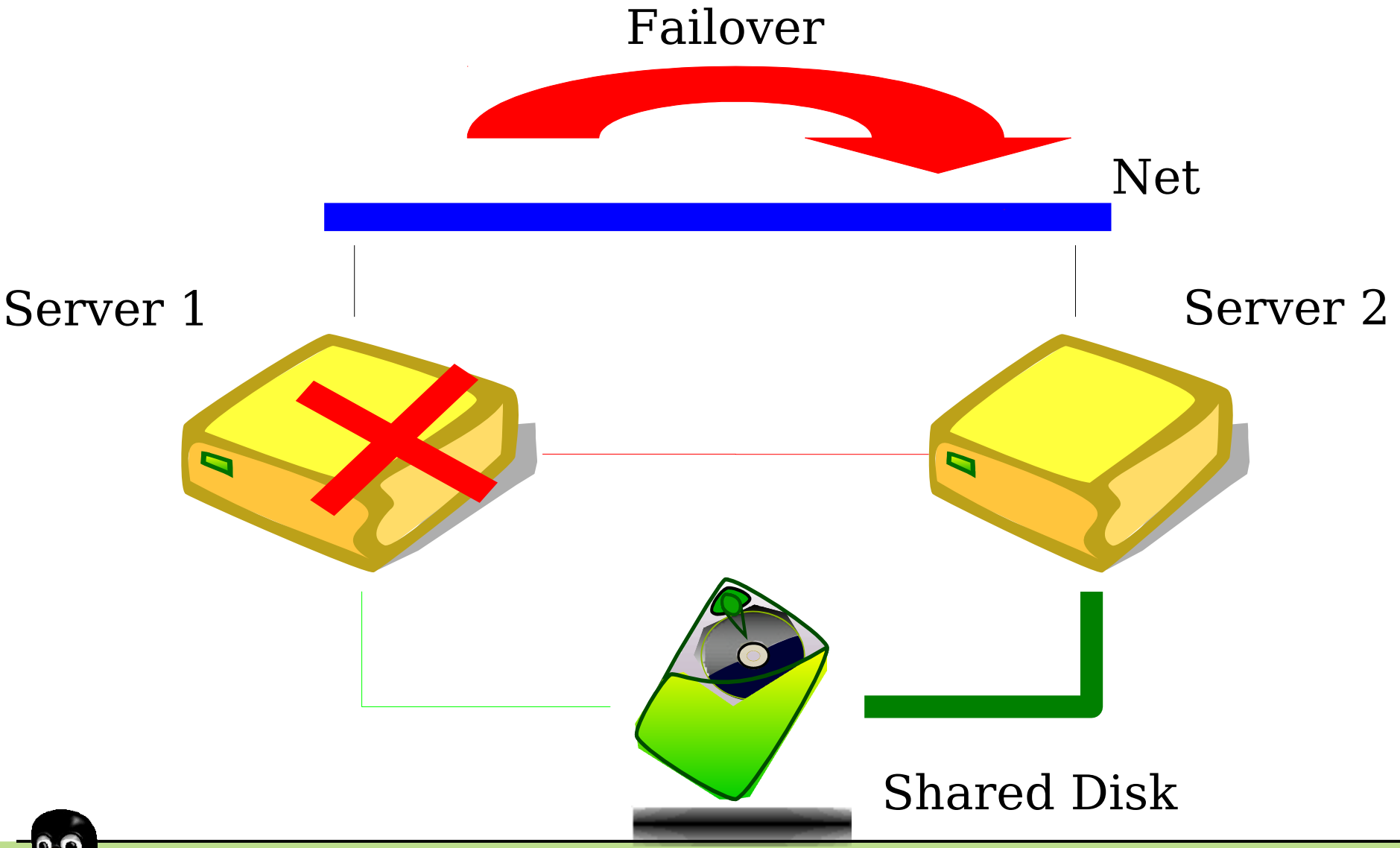
Availability vs. Reliability

- Availability and reliability are related each other but they are different
- Example: a commercial web site goes down for only one minute every four hours, i.e. every 240 minutes. Availability is:

$$A = \frac{240-1}{240} = \frac{239}{240} = 99{,}583\%$$

- It seems a reasonably high availability, but the reliability is low if the down times occur at a critical time during transactions!

High Availability clusters



New systems design

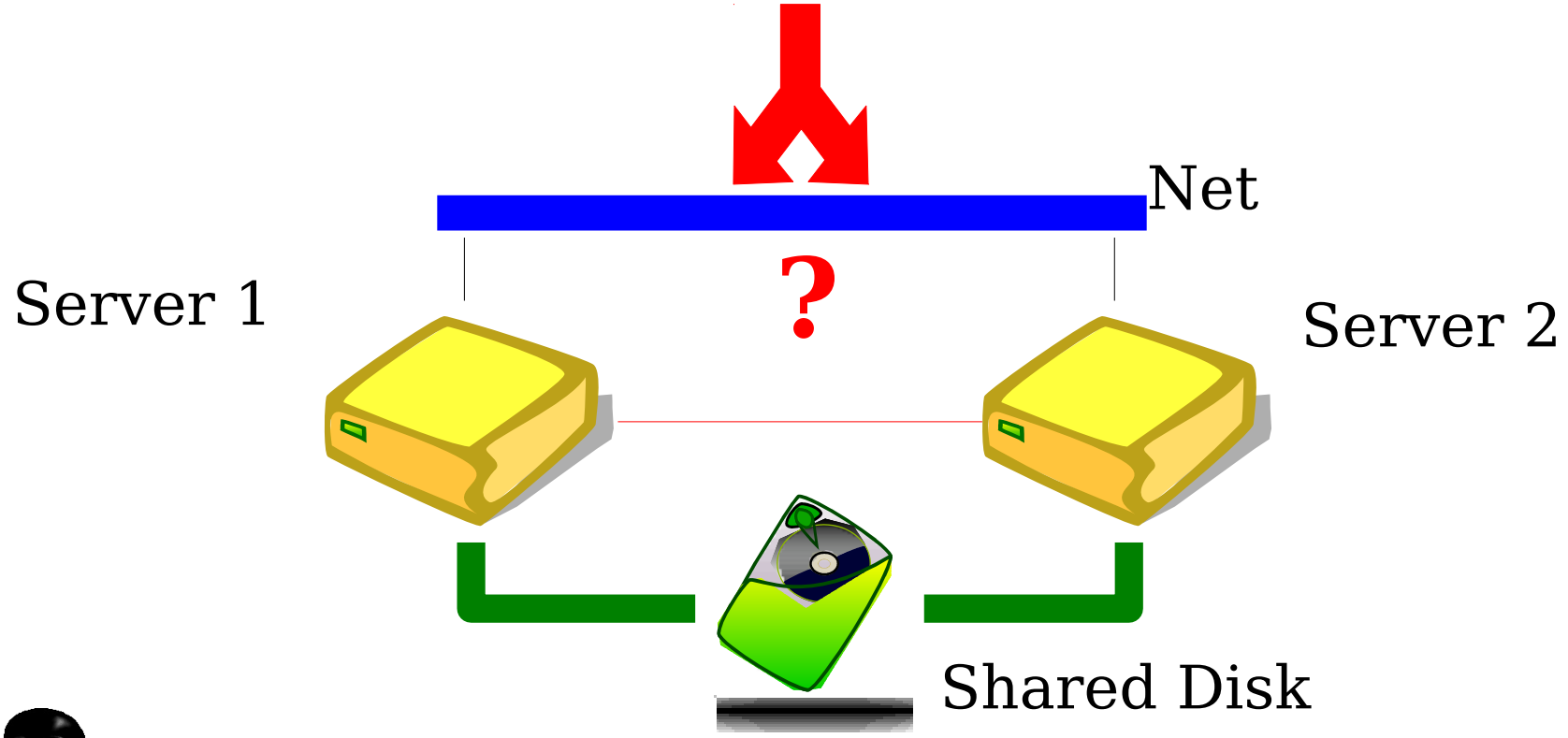
- Data sharing between cluster members
- Design systems without Single Point of Failure (SPOFs)
- HA Clusters introduce new concepts and challenges:
  - Split-Brain
  - Quorum
  - Fencing
- It's very important to know WHERE applications run and where they could do it
- Very important understand and forecast availability

Availability analysis

- With this kind of analysis we can estimates some parameters
- Mean Time To Repair (MTTR): average time it takes for the system to recover
- Mean Time To Failure (MTTF): average time it takes for the system to fail
- For a given configuration we can estimate the ratio between MTTR and MTTF
- If we fix the MTTF we can claim the MTTR (for SLA for example) to have a given availability

Split-brain

- Communications failures can lead to serious problems
- If each cluster member try and take control of the cluster, then it's called a split-brain condition



Quorum

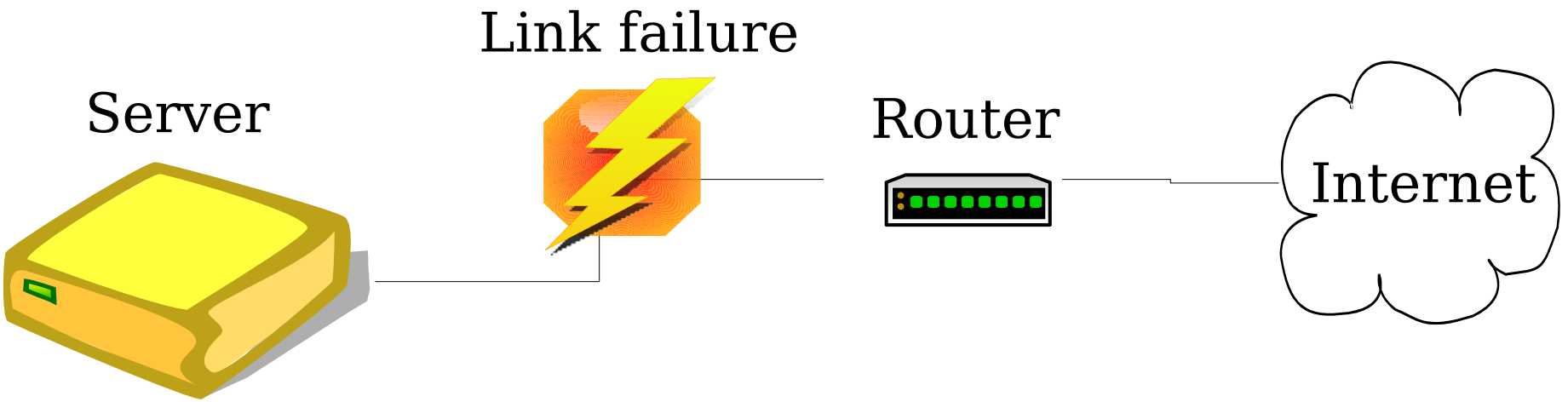
- Quorum can avoid split brain for many kinds of failures
- Typically one tries to make sure that only one node can be active
- Quorum is the term used to refer to methods for ensuring only one active node
- Most common kind of quorum is voting – and only a node with more of n/2 nodes can be the master of the cluster
- This doesn't work, however, very well for 2 nodes

Fencing

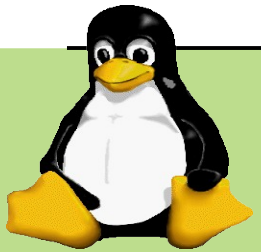
- Fencing tries to put a fence around an errant node or nodes to keep them from accessing cluster resources
- This way one doesn't have to rely on correct behavior or timing of the errant node
- Main techniques:
  - Turn off the power of the other nodes
  - Fiber channel switch lockout

Single Point of Failure (SPOF)

- A single point of failure is a component whose failure will cause the failure of the entire system:



- In this example, after the link failure our server won't be reachable anymore from the outside world!



Redundancy

- How we can eliminate SPOFs?
- The answer is Redundancy!
- Most SPOFs are eliminated by managed redundancy
- Clustering is a good way of providing and managing redundancy
- Find out all SPOFs in a system and solve them with redundacy can be very difficult!

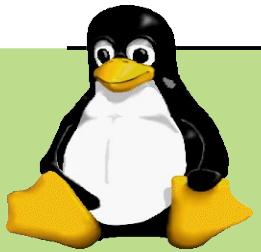
Redundant links

- Intra-cluster communication is critical to HA system operation
- Provide mechanisms for redundant internal communication
- External communications is usually essential to provision of service
- Redundancy through routing tricks

Redundant data

- When a service comes up on another node is usually needed to have the same data of the node down
  - Clustering filesystems (as OCFS2)
  - Shared redundant disk (RAID systems)
  - Mirroring systems (RAID-1) across the network (as DRBD)
- Some environments can live with less “precise” replication methods – rsync for example
- Not always needed

Building HA Systems
Linux-HA framework



Linux-HA history

- Open source HA project - providing sophisticated fail over and restart capabilities for Linux (and other OSes)
- In existence since 1998
- More than 30000 mission-critical clusters in production since 1999
- Active and open development community
- Wide variety of industries, applications supported
- Shipped with most Linux distributions
- No special hardware requirements; no kernel dependencies, all user space

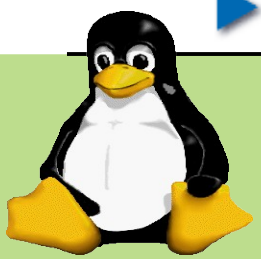
Linux-HA features

- Can use serial, UDP bcast, mcast, ucast communication
- Fails over on any condition: node failure, service failure, IP connectivity, arbitrary criteria
- Configuration and monitoring GUI
- Supports n-node clusters (more or less 16)
- Built-in resource monitoring
- Support for the OCF resource standard
- Sophisticated dependency model with rich constraint support (resources, groups, incarnations, master/slave)
- XML-based resource configuration
- Multi-state (master/slave) resource support
- Split-site cluster support with quorum daemon

Only with Release 2

Linux-HA releases

- You'll find R1 (release 1) and R2 (release 2) terminology, actually same sw release but different configurations
- R2 it's more complete and flexible
- R2 has got a difficult xml file to write called CIB (Cluster information base)
- R1 it's the best choice if you have a simple environment and constraints
- Three configuration file:
  - ha.cf
  - authkeys
  - haresources for R1 and cib.xml for R2



ha.cf file

- Main settings:
  - Udpport: port used to send heartbeat messages
  - baud and serial: baud rate and serial communication port
  - bcast, mcast, ucast: addresses used to send heartbeat messages
  - auto_failback: we can decide if after a takeover resources have to return to “home”
  - watchdog: heartbeat process can manage watchdog device to achieve more reliability

authkeys file

- Heartbeat messages usually can be send in trusted network
- We can have different level of authentication:
  - CRC: not authentication actually but only packet check error, for trusted networks
  - MD5: good algorithm to use to achieve authentication
  - SHA1: best option, more difficult to crack
- Authkeys file should be 0600 mode

haresources file

- In this file we can claim the resources and the owner of the resources
- For each line <node-name> <service1> .... <serviceN>
- Services are the names of scripts under `/etc/ha.d/resources.d` folder
- To pass parameters to the script we can use the operator `::`
- For example, to claim the ownership of a service IP address with subnet and broadcast address:

```
IPaddr2::135.9.216.3/28/eth0/135.9.216.12
```

Resource types in R2

- Release 2 supports OCF (Open Cluster Framework) standard
- Release 2 supports *resources* which can be any of the following:
  - Primitive: OCF, heartbeat-style, or LSB resource agent scripts
  - Clones – need "*n*" resource objects – somewhere
  - Groups – a group of resources with a co-location and linear ordering constraints
  - Multi-state (master/slave): designed to model master/slave (replication) resources (DRBD)

CIB

- It has to be owned by haclient:hacluster
- It should be mode 0600
- If we have an R1 configuration we can convert our `haresources` file in a `cib.xml` with the script `haresources2cib.py`
- CIB is described here:

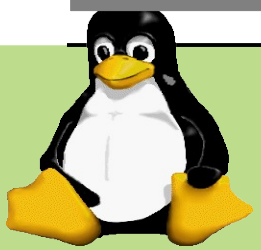
`http://hg.clusterlabs.org/pacemaker/dev/file/tip/xml/crm-1.0.dtd`

Primitive resources

```
<primitive id="WebIP" class="ocf" type="IPaddr2" provider="heartbeat">
<operations>
<op id="1" name="monitor" interval="5s" timeout="3s"/>
</operations>
 <instance_attributes>
  <attributes>
    <nvpair id="someid"
        name="ip"
        value="135.9.216.3"/>
     <nvpair id="someid2"
        name="nic"
       value="eth0"/>
    <nvpair id="someid2"
        name="cidr_netmask"
        value="255.255.255.240"/>
    <nvpair id="someid2"
        name="broadcast"
        value="135.9.216.12"/>
  </attributes>
 </instance_attributes>
</primitive>
```

How long to wait before declaring it dead

How long to wait for an action to complete



Resource Groups

- Resource Groups provide a simple method for creating ordering and co-location dependencies
- Each resource object in the group is declared to have linear *start-after* ordering relationships
- Each resource object in the group is declared to have co-location dependencies on each other
- Resources are stopped in the reverse order they're started

Resource clones

- Resource Clones allow one to have a resource which runs multiple (“*n*”) times on the cluster
- This is useful for managing
  - load balancing clusters where you want “*n*” of them to be slave servers
  - Cluster filesystems
  - Cluster Alias IP addresses
- Clone constructs can contain either primitives or groups

Multi state resources

- Resources usually have two different states started and stopped
- Multi-state resources can have more than two states:
  - Stopped
  - Running master
  - Running slave
- Modelling replication resources like DRBD

Heartbeat

- Simple idea: send a message every so often to say “hey, I'm alive!”
- Main settings in ha.cf are:
  - keepalive: how long to wait before send another message
  - deadtime: when this time expires the other node is dead
  - warntime: when this time expires a log entry with a warning is created
  - initdead: it is used instead of deadtime the first time heartbeat starts. It is useful if we have some initial latency in the network

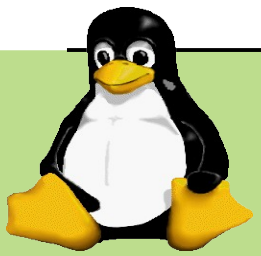
ipfail

- Ipfail (or pingd if we use R2) is an useful tool to detect lost of connectivity with the external network
- Main settings in ha.cf are:
  - ping: we can specify an IP address to test our connectivity
  - ping_group: we can even use a group of node to test it, if at least one member of the group if reachable we haven't lost it
  - deadping: when this time expires the ping node is not considered not reachable
- Ping node are *pseudo*-member of the cluster
- Don't use a cluster member as ping node!

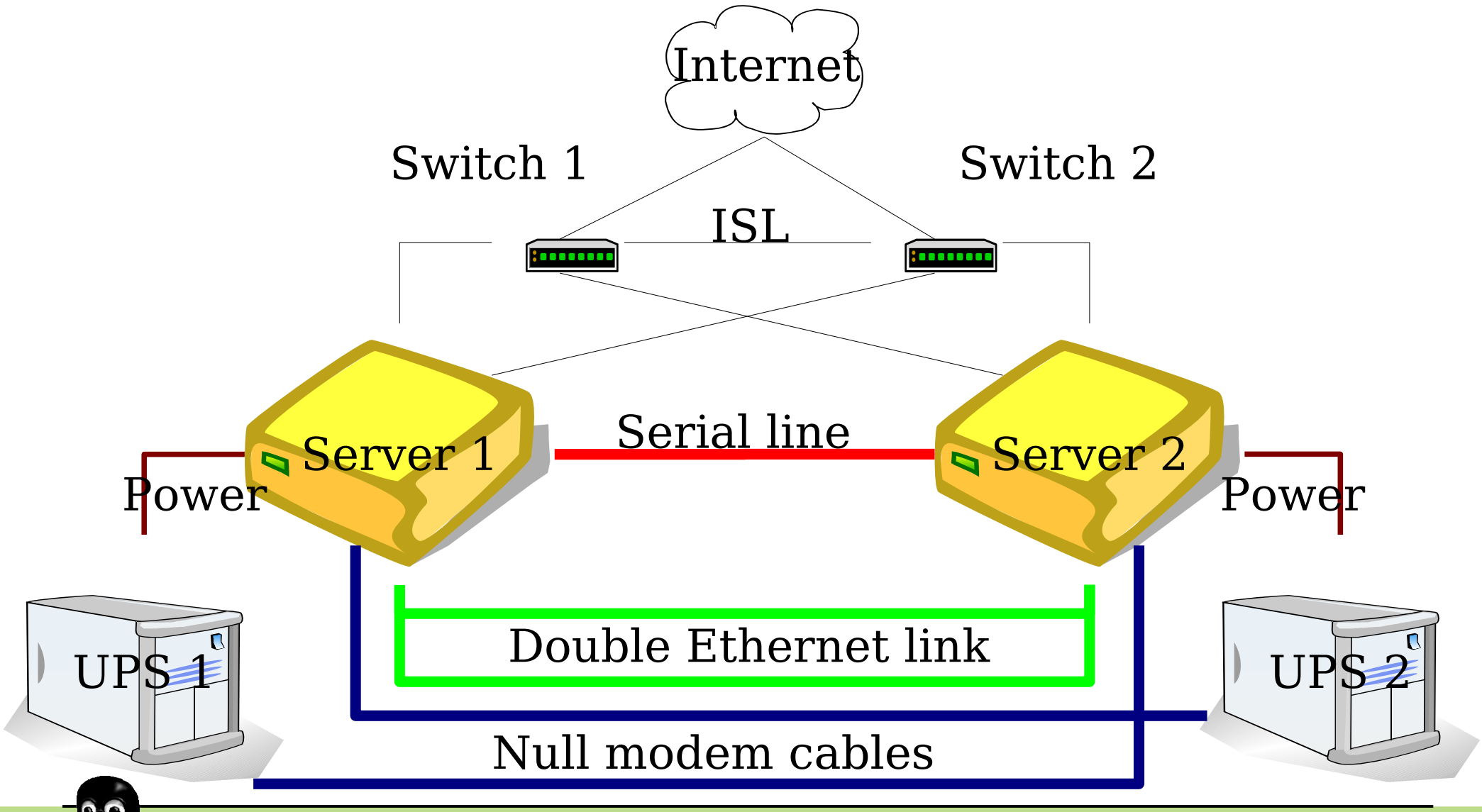
Stonith

- STONITH is a fencing technique
- STONITH turn off the other node, so we can be sure that it's dead!
- Main settings in ha.cf are:
  - stonith <device_type> <config_file>: for basic configuration, if we have only one stonith device in the cluster
  - stonith_host <hostfrom> <device_type> <params>: if we have more than one device in the cluster
- Examples:

```
stonith baytech /etc/ha.d/conf/stonith.baytech
stonith_host * baytech 10.0.0.3 mylogin mypassword
```



Putting all together



Quorum server

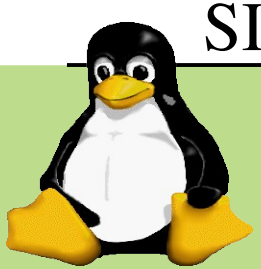
- Remote communications are never as reliable as local communications
- Fencing techniques (as STONITH) require highly reliable communications
- Split-site clusters from fencing point of view are very problematic
- How to solve it?
  - Quorum without fencing must be used instead

How it works

- Each cluster member is connected with a node called Quorum server
- One Quorum server for each cluster
- Quorum server provides an extra quorum vote and it is not a cluster member
- Quorum server does not require special networking
- Reliability of Quorum server and links to it are important
- If the communications between sites goes down sites contact Quorum server
- Quorum server gives quorum to only one member and only one will be the master

Using Quorum

- Two world wide sites called SITE1 and SITE2 and one Quorum server:
  - SITE1 goes down
  - Quorum server supplies extra quorum vote
  - Cluster retains quorum
  - SITE2 continues to provide service
- If only the Quorum server goes down, cluster will be still up
- If quorum server and SITE1 are down, SITE2 will be up but without quorum no service! **Single fault** only!
- Quorum can be overridden manually to force service at SITE2



References

- Linux-HA site
  `http://linux-ha.org/`
- Tutorials
  `http://www.linux-ha.org/HeartbeatTutorials`
- Download Page
  `http://www.linux-ha.org/DownloadSoftware`

Building HA Systems
DRBD

DRBD

- For many high-availability system data redundancy in the cluster is mandatory
- Mains approaches:
  - Disk sharing
  - Network mirroring
- DRBD is a free tool to do mirroring of data trough the network

DRBD features (1)

- Two way replication and starting from 8.3.0 even three way
- Fully synchronous, memory synchronous or asynchronous modes of operation
- Shared secret to authenticate the peer upon connect
- Bandwidth of background resynchronization tunable
- Automatic recovery after node, network or disk failures
- Efficient resynchronization, only blocks that were modified during the outage of a node
- Short resynchronization time after the crash of an active node, independent of the device size

DRBD features (2)

- Automatic detection of the most up-to-date data after failure
- Integration scripts for use with Heartbeat
- Dual primary support for use with GFS/OCFS2
- Configurable handler scripts for various DRBD events
- Online data verification
- Optional data digests to verify the data transfer over the network
- Heartbeat integration to outdate peers with broken replication links, avoids switchovers to stale data

drbd.conf

```
global{......}
common{......}
resource r0{
   handlers{......}
   startup{......}
   disk{......}
   net{......}
   syncer{......}
   on foo1{......}
   on foo2{......}
}
......
```

- Main section
- Common options section
- Per resource section
- Action handlers
- Startup options
- Disk options
- Net options
- Synchronization options
- Nodes sub-sections

Global section

- minor-count
  DRBD module minor-count. Default is 32
- dialog-refresh
  The user dialog counts and displays the seconds it waited so far. You might want to disable this if you have a server with limited logging capacity. Default is 1 second
- disable-ip-verification
  To disable one of drbdadm's sanity check
- usage-count
  To participate in DRBD's online usage counter

Common section

- The common section can have all the sections a resource can have but not the node section
- The common section must precede all resources
- All resources inherit the settings from the common section
- Settings in the resources have precedence over the common setting

Resource section

- It has got several sub-sections: handlers, syncer, net.....
- It has got only one option: protocol
- We can use three type of *protocol*:
  - A: write IO is reported as completed, if it has reached local disk and local tcp send buffer. For high latency networks
  - B: write IO is reported as completed, if it has reached local disk and remote buffer cache. For most cases
  - C: write IO is reported as completed, if we know it has reached both local and remote disk. For critical transactional data

Handlers section (1)

- Pri-on-incon-degr
  What should be done in case the node is primary, no connection and has inconsistent data
- Pri-lost-after-sb
  The node is currently primary, but lost the after split brain auto recovery procedure. As as consequence it should go away
- local-io-error
  In case of *on-io-error* (see disk section) option equals to *call-local-io-error*, this script will get executed in case of a local IO error

Handlers section (2)

- outdate-peer
  Commands to run in case we need to downgrade the peer's disk state to "Outdated". (Application provided by heartbeat!)
- pri-lost
  The node is currently primary, but should become sync target after the negotiating phase. Alert someone about this incident
- split-brain and out-of-sync
  Notify in case split brained or out-of-sync
- before-resync-target and after-resync-target
  These two handlers can be used to snapshot sync-target devices before for the time of the resync

Startup section

- wfc-timeout
  Wait for connection timeout
- degr-wfc-timeout
  Wait for connection timeout if this node was a degraded cluster. In case it is rebooted, this value is used
- wait-after-sb
  By setting this option you can make the init script to continue to wait even if the device pair had a split brain situation and therefore refuses to connect
- become-primary-on both
  In case of use GFS/OCFS2

Disk section

- on-io-error
  If the lower level device reports io-error we can report the io-error to the upper layers or call the script “local-io-error" or drop the backing storage and continues in disk less mode
- fencing
  Fencing policy
- size
  In case we only want to use a fraction of the available space
- no-disk-flushes and no-md-flushes
  To disable disk flushes

Net section (1)

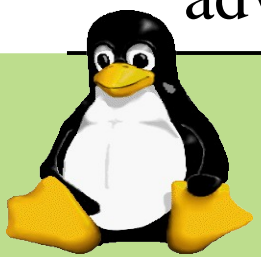
- timeout
  Connection timeout time
- Connect-int
  Connection interval between two connection tries
- ping-int
  Interval between two keepalive messages
- ping-timeout
  Keepalive timeout time
- cram-hmac-alg and shared-secret
  Peer authentication

Net section (2)

- If after a split-brain situation the nodes of the cluster see each other again we can decide the auto recovery policy with three different option (*after-sb-0pri*, *after-sb-1pri*, *after-sb-2pri*)
  - We could decide to discard older (or younger) primary or discard always a given node
- We can decide what to do in case the outcome of the resync decisions is incompatible to the current role assignment in the cluster
  - We could call a script to reboot the node and make it secondary

Syncer section

- Rate
  To limit the bandwidth used by the resynchronization process
- After
  We can decide to sequentialize the resynchronization process over the resources
- Al-extents
  Size of "*hot area*". Each "extent" is 4MB. With 257 extents and with a link of 10MB/s we can have a resync in less then 2 minutes
- Cpu-mask
  Sets the CPU affinity mask of DRBD's threads. It's useful for advanced tuning



Node section

- Device
  DRBD's device file (`/dev/drbdN`).
- Disk
  Device file of the block device (`/dev/mtdblockN`)
- Address
  IP address of the node and the port of DRBD listens (`<IP ADDRESS>:<PORT>`)
- Meta-disk
  DRBD uses some meta-data to store information. With this option we can deploy these information in a specific partition.

DRBD Meta-data

- We have four choices to deploy drbd's meta-data in our disk:
  - `meta-disk internal`
  - `meta-disk /dev/<disk> [index]`
  - `flexible-meta-disk internal`
  - `flexible-meta-disk /dev/<disk>`
- Internal means that the last part of the backing device is used to store the meta-data
- We can use a single block device to store meta-data of multiple DRBD devices. In this case we use the index
- flexible-meta-disk is an useful option to use with LVM

Meta-data size

- We can figure out the size of meta data with this formula:

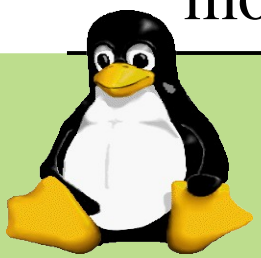
$$M_s = \left\lceil \frac{C_s}{2^{18}} \right\rceil * 8 + 72$$

- Where M_s and C_s are in sectors. We can even figure out the size in MB with some approximation:

$$M_{MB} < \frac{C_{MB}}{32768} + 1$$

DRBD Tools

- DRBD provides a kernel module and several management tools as drbdadm, drbdsetup, drbdmeta
- DRBD module in kernel mainline starting from 2.6.33
- Drbdadm is the high level tool of suite. It is to drbdsetup and drbdmeta what ifup/ifdown is to ifconfig
- Drbdadm reads its configuration file and performs the specified commands by calling the drbdsetup and/or the drbdmeta program
- Use drbdmeta and drbdsetup only when really needed!
- DRBD provides a proc file `/proc/drbd` very useful to monitor the status



DRBD Setup (1)

- Create device metadata. This step must be completed only on initial device creation. It initializes DRBD's metadata:

```
drbdadm create-md resource
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initialized bitmap
New drbd meta data block sucessfully
created.
success
```

DRBD Setup (2)

- Attach to backing device. This step associates the DRBD resource with its backing device:
  `drbdadm attach resource`
- Set synchronization parameters. This step sets synchronization parameters for the DRBD resource:
  `drbdadm syncer resource`
- Connect to peer. This step connects the DRBD resource with its counterpart on the peer node:
  `drbdadm connect resource`
- In alternative to these three steps we could use:
  `drbdadm up`

DRBD Setup (3)

- Now we have still got an inconsistent state on both nodes
- We need the first resync operation:

```
drbdadm -- --overwrite-data-of-peer primary resource
```

- After that operation DRBD is now fully operational

DRBD proc file

- With the drbd proc file we can monitor the status of the connection, the role of each node and the disk status:

```
cat /proc/drbd
version: 8.3.0 (api:88/proto:86-89)
 0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:0 nr:8 dw:8 dr:0 al:0 bm:2 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
 1: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

DRBD and Heartbeat

- Integration of DRBD into Heartbeat is very simple in R1 style, only one line in the ha.cf file:

```
node1 drbddisk::r0 Filesystem::/dev/drbd0::/share::jffs2
```

- R2 style is meanly more difficult to setup, but a very good starting point is the DRBD User's guide
  `http://www.drbd.org/users-guide/s-heartbeat-crm.html`
- As general rule it is suggested to set DRBD's timeouts smaller then the Heartbeat ones

DRBDLinks

- DRBDlinks is a handy tool for managing symbolic links for filesystem mounts
- It is useful with filesystem replication (DRBD) or shared disk arrangements
- You need one drbdlinks resource for each filesystem you want to manage with it
- It is currently only available as a Heartbeat classic R1 style resource (not yet as an OCF R2 resource)
- We can share data without copy information in our mirrored partition!

References

- DRBD Project Home page
  `http://www.drbd.org`
- DRBD Users's Guide
  `http://www.drbd.org/users-guide`
- DRBDLinks
  `http://tummy.com/Community/software/drbdlinks/`

Building HA Systems
Watchdog

Watchdog

- When the system hangs we need help
- Software management of critical system blocks is not possible
- Special hardware comes in help
- Watchdog triggers an hard reset after a specified timeout
- To avoid system reboot a software *kicker* reset the timer writing in some cpu register
- In Linux we have the device file `/dev/watchdog` used by the watchdog daemon

Watchdog features (1)

- Periodical system checks about:
  - Process table size
  - Free memory
  - Files accessibility
  - Files changed within a given interval
  - Average work load
  - File table overflow
  - Process running
  - Network reachability
  - System temperature
  - User defined command to do arbitrary tests

Watchdog features (2)

- If only one of the previous test fails watchdog will cause a shutdown
- Should any of these tests except the user defined binary last longer than one minute the machine will be rebooted, too
- We can set a user-defined binary in case of a problem instead of shutting down the system
- We can set an email address to notify that the machine is being halted or rebooted

Watchdog.conf (1)

- Interval
  Set the interval between two writes to the watchdog device
- Logtick
  logtick allows adjustment of the number of intervals skipped before a log message is written. Very useful to avoid watchdog flooding messages in the syslog
- Max-load-1, max-load-5 and max-load-15
  Max load average allowed in the last one, five and fifthteen minutes
- Min-memory
  Min memory free in the system in number of pages

Watchdog.conf (2)

- Max-temperature
  Max temperature of the system
- Watchdog-device
  Watchdog device file
- File
  Set file name for file mode. This option can be given as often as we like to check several files
- Change
  Set the change interval time for file mode
- Pidfile
  Set pidfile name for process test. This option can be given as often as you like to check several processes

Watchdog.conf (3)

- Ping
  Set IP address for ping mode. This option can be used more than once to check different connections
- Interface
  Set interface name for network mode. This option can be used more than once to check different interfaces
- Test-binary
  Execute the given binary to do some user defined tests
- Test-timeout
  User defined tests may only run for <timeout> seconds. Set to 0 for unlimited

Watchdog.conf (4)

- Repair-binary
  Execute the given binary in case of a problem instead of shutting down the system
- Admin
  Email address to send admin mail to
- Realtime
  If set to yes watchdog will lock itself into memory so it is never swapped out
- Priority
  Set the schedule priority for realtime mode

Watchdog kernel support

- Under device drivers we find *Watchdog* Timer Support
- If our cpu has got a watchdog device we can enable the driver
- If our cpu doesn't have watchdog support we could use the software watchdog
- Software watchdog is a software simulated watchdog, it triggers a reboot when a timer expires
- We can enable the option *NOWAYOUT* to keep up the timer running even when the device file is closed after the opening

Softdog (1)

- Softdog is a very low priority demon that improve the reliablity of watchdog demon
- If watchdog demon runs with low priority but the system is really loaded, it could reboot because the demon hadn't got a scheduler slot and the timer expired
- Setting a realtime priority for watchdog ensure that it will have a chance to run, but what about the other applications?
- Softdog runs with very low priority and it waits for data on a local socket

Softdog (2)

- You can configure the watchdog demon to use the script testsoftdog as test binary
- It sends to Softdog a random value and it waits for a new value incremented by one
- With this schema you can be sure that every application with a priority between watchdog demon priority and the lower system priority has got a chance to run
- You can find Softdog on my site: `http://www.webalice.it/firefox82/softdog.tar.gz`

Building HA Systems
Mon

Mon

- Mon is a tool for monitoring the availability of services and applications
- Used for fault detection and alert management
- It can send alerts to system administrator when something goes wrong (e.g. a link is down)
- It can monitor an application and trigger a resource fail-over in a high-availability configuration!

Mon features

- Portable because written in Perl on several OS as Linux, Solaris, BSD, Cygwin (Windows)
- Simple and extensible design
- Can monitor anything, no clients or agents required
- Full community support
- Community provides custom monitors, alerts, and other tools.

Mon design

- Simple to add alerts and monitors
- Simple way of cross-connecting tests and alerts
- Simple way of gathering data for report generation
- General-purpose, platform-agnostic, if you can test it with software, you can monitor it
- Five main components: Servers, Traps, Monitors, Alerts and Clients

Mon architecture

- Server
  Schedules and executes monitors (tests), handles traps, alerts, clients, logs
- Clients
  Query and control the server, show reports
- Monitors
  Communicate with monitored systems via HTTP, SNMP, etc.
- Traps
  Send notifications to other systems (Mon systems or otherwise)
- Alerts
  Perform actions on failures, page, email, trouble ticket, corrective action (HA fail-over)

Mon server

- Schedule tests
  - Run monitors when necessary and gather output and exit status
- Accept remote traps
- Serve clients
  - Deliver operational status
  - Accept control commands
- Manage Alerts
  - Suppress repetitive alerts
  - Alert only during specified time periods
  - Evaluate dependencies

Mon configuration file

- Configuration file called mon.cf under etc folder (default)
- It specify the linking of monitors to alerts
- Defines what is to be monitored and how:
  - Monitors used
  - Hosts and services monitored
- Defines when alerts happen:
  - On which failure
  - On which success after a failure transition
  - Frequency
  - Time of day

Mon monitors (1)

- Test the condition of a service:
  - Usually one service test per monitor.
  - Tests are user-definable.
  - SNMP, HTTP, SMTP, ICMP echo and so on.
  - Application-level tests possible
- Report summary and detailed results
- Exit reporting success/failure

Mon monitors (2)

- Written in an arbitrary language
- May call third-party software
- Independent from the Mon server
- Invoked as separate processes
- Monitor life-cycle: start, test, report and exit
- Very simple to write

Built-in monitors

- Monitors provided together with Mon:
  - reboot, freespace, pop3, smtp, ftp
  - processes, dialin, msql, mysql
  - radius, http, RAID, tcp, dns, telnet
  - imap, nntp, rpc, trace, file change
  - ldap, ntp, traceroute, syslog, fping, lpd, ping

Adding a monitor

- Each monitor receive a list of items
- Some standard env variables are set MON_LOGDIR, etc.
- It performs tests on items
- First line of output must be the summary line
- Remaining lines are the detail
- Exit status of zero (success) or nonzero (failure)

Monitor example

- Monitor several hosts with a simple ping:

```
PING="ping -c 1"
failed=""
for hosts in "$@"
do
    if ! $PING $hosts >/dev/null 2>/dev/null; then
        if [ "$failed" = "" ]; then
            failed="$hosts"
        else
            failed="$failed $hosts"
        fi
    fi
done

if [ "$failed" != "" ]; then
    echo "$failed"
    exit 1
fi

exit 0
```

Mon traps

- Traps are notifications sent to a Mon server from an
  - external entity
  - another Mon server
- Contain the same information as passed by monitor scripts
  - summary
  - detail
  - exit status
- Allows distributed Mon agents to send their status to a centralized Mon server

Mon alerts

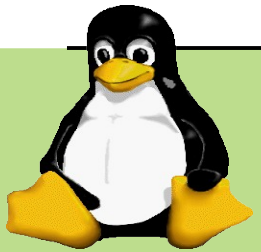
- Report the failure status detected by a monitor
- Independent from the Mon server
- Accept input from the Mon server
- Invoked as separate processes
- Written in any language
- Simple to write

Built-in alerts

- E-Mail
- Log on a file
- SNPP (alphanumeric paging via TCP/IP)
- Quick page (alphanumeric paging via modem and TAP/IXO)
- Trap to other Mon server
- IRC

Alerts management

- Alert decision logic in the server
- Full control over periodic alerts
  - alertafter [*num*] [*interval*]: send an alert only after *num* failure or after *num* failure within *interval* or only after *interval*
  - alertevery: send an alert over a fixed period
  - numalerts: max number of alerts
- Dependencies:
  - We can create a dependency between alerts in order to trigger only the first one and avoid the others
  - Dependencies are Perl expressions



Adding a new alert

- An alert receives some options supplied by server and from the config file
- The first line of stdin is the summary, rest is detail
- Perform whatever action desired even a take over!

Alert example

- Trigger a take over via heartbeat command:

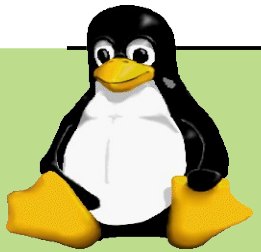
```
#!/bin/bash
hb_standby="/usr/lib/heartbeat/hb_standby"

if $hb_standby >/dev/null 2>/dev/null; then
   exit 0
else
   exit 1
fi
```

Alert template

```
#!/usr/bin/perl
use Getopt::Std;
getopts ("s:g:h:t:l:u");
$summary=<STDIN>;
chomp $summary;
$t = localtime($opt_t);
($wday,$mon,$day,$tm) = split (/\s+/, $t);
print <<EOF;
Alert for group $opt_g, service $opt_s
EOF
print "This alert was sent because service was restored\n"
    if ($opt_u);
print <<EOF;
This happened on $wday $mon $day $tm
Summary information: $summary
Arguments passed to this script: @ARGV
Detailed information follows:

EOF
while (<STDIN>) {
    print;
}
```



Mon clients

- Mon protocol, registered port 2583 with IANA
- Easy Perl interface, Mon::Client
- Get operational status of things monitored
- Disable/enable monitoring and alerting
- Acknowledge alerts sent
- Allows for many reports

Configuring Mon (1)

- Send an email when any web servers become unpingable:

```
hostgroup servers www ftp pop3 smtp

watch servers
   service fping
      monitor fping.monitor
      interval 1m
      period wd {Sun-Sat}
         alert mail.alert myemail
         alertevery 24h
         upalert mail.alert myemail
```

Configuring Mon (2)

- Another more complicated example:

```
watch serverfoo
    service ping
        monitor ping.monitor
        interval 2m
        period label1: wd {Sun-Sat}
            alert mail.alert myemail
            alertevery 12h
            upalert mail.alert myemail
        period label2: wd {Sun-Sat}
            alert mail.alert myspecialemail
            alertevery 24h
            alertafter 5 10m
        period label3: wd {Mon-Fri} hr {9am-6pm}
            alert mail.alert staffemail
            alertevery 4h
    service ftp
        monitor ftp.monitor
        interval 5m
        depend SELF::ping
        period wd {Sun-Sat}
            alert mail.alert myemail
            alertafter 10m
            numalerts 2
```

References

- Download Mon
  `http://www.kernel.org/software/mon/`
- Wiki Mon
  `http://mon.wiki.kernel.org/index.php`
- Mon Mailing list
  `mon@linux.kernel.org`

Kernel and High Availability

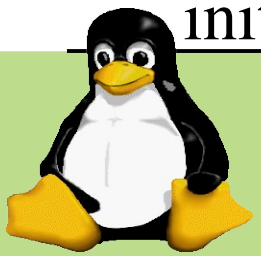
Kernel Patching

Ksplice

- Kernel bugs and security fixes are constantly detected and corrected
- To apply these updates the whole system must be rebooted
- Rebooting is a disruptive process, it increase the system downtime
- Ksplice makes it easy to apply Linux updates seamlessly, with no downtime or in other words without reboot!

Ksplice features (1)

- Works with Linux 2.6.x kernel versions (tested from 2.6.8 to 2.6.28-rc6)
- Works with CONFIG_KALLSYMS, but it can apply most patches without that option
- Can handle patches to kernel modules and kernel assembly code
- Can handle patches containing symbols that are not in the kernel symbol table (symbols can be discovered from running code)
- It cannot manage patches (without add a bit of code) with semantic changes: add/remove global structure fields, initialization change of global data and so on

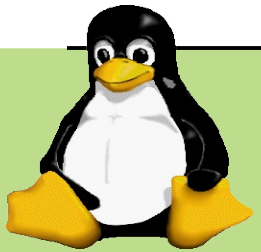


Ksplice feature (2)

- Works best if used with the exact compiler used to compile the original kernel
- When the original toolchain is not available, Ksplice will check out the differences behavior between the toolchains and it will abort the upgrade if necessary
- Architecture currently supported:
  - x86-32
  - x86-64
  - ARM

How it works (1)

- Ksplice replaces all of the functions that a patch changes
- It replaces a function by linking the patched version of the function into the kernel's address space and by causing all callers of the original function to invoke the patched version
- It accepts as input the original kernel source code and a source code patch
- Ksplice performs two kernel builds and looks at how the resulting ELF object files differ:
  - No C parsing and no compiler dependency
  - No inline functions substitution problem



How it works (2)

- It extracts the changed functions and puts them into an object file
- Then it creates a kernel module by combining this object file with generic code for performing hot updates
- Now we have our *patch module*!
- The updates is not volatile and it persists over a reboot because it is a normal module
- In user-space there are four tools to manage the kernel updates with ksplice: `ksplice-create`, `ksplice-apply`, `ksplice-view` and `ksplice-undo`.

Patching the kernel (1)

- Create a subdirectory ksplice under our kernel source code folder:
```
mkdir /linux-source/ksplice
```
- Copy our config and System map files in the ksplice subdirectory:
```
cp config-2.6.28 /linux-source/ksplice/.config
cp System.map-2.6.28 /linux-source/ksplice/System.map
```
- If we are using a distribution-packaged kernel, we should also add a link to our kernel headers:
```
ln -s /lib/modules/2.6.28/build /linux-source/ksplice/build
```

Patching the kernel (2)

- Create the patch module and load it (as root):

```
#ksplice-create --id=1234 \
--patch=security.patch /linux-source
Ksplice update tarball written to ksplice-
1234.tar.gz

#ksplice-apply ./ksplice-1234.tar.gz
Done!
```

Patching the kernel (3)

- We can view the patch modules loaded and remove them:

```
# ksplice-view
1234
# ksplice-view --id=1234
Ksplice id 1234 is present in the kernel and is
applied.

Here is the source code patch associated with this
update:
...........
# ksplice-undo 1234
# dmesg | tail -n1
ksplice: Update 1234 reversed successfully
```

References

- Ksplice Home site:
  `http://www.ksplice.com`

Kernel and High-Availability

Kexec

New Kernel with kexec

- When we upgrade an application we don't need to reboot the overall system but we MUST reboot the system if we change the kernel
- Kernel upgrade could be mandatory due to security bugs
- Kexec makes it possible to call a new kernel, without rebooting and going through the BIOS / firmware
- Kexec can reduce the system downtime

Using kexec (1)

- Kexec system call is supported in `alpha, arm, x86, ia64, mips, powerpc, powerpc64, s390, x86_64, superH`
- We need some tools called kexec-tools in user space
  See http://www.kernel.org/pub/linux/kernel/people/horms/kexec-tools/kexec-tools.tar.gz
- We need a kernel compiled with CONFIG_KEXEC option enabled under *Processor type and features* menu

Using kexec (2)

- To load the new kernel
  `kexec -l <kernel-image> --append=<command-line-options> --initrd=<initrd-image>`
- Now jump to the new one
  `kexec -e`
- We can call kexec without an option parameter. In that case, kexec loads the specified kernel and then invokes shutdown
- Shutdown scripts will call `kexec -e` just before actually rebooting the machine
- That way, the machine does a clean shutdown including all shutdown scripts

Kernel and High-Availability
Kernel Errors

Kernel Panic

- There are two kind of errors that can happen in the kernel:
  - Kernel oops!
    Soft error condition, the system usually will be still up and running.
  - Kernel panic
    Critical error condition, the system will be rebooted after a configurable timeout.
- In order to grant high availability and stability, it is useful to manage oops like kernel panic

Kernel Panic tuning

- First of all we set the timeout (five seconds in this example)
  ```
  echo 5 > /proc/sys/kernel/panic
  ```
- After that, we tell to the kernel that we want always manage oops error as panic error
  ```
  echo 1 > /proc/sys/kernel/panic_on_oops
  ```
- We can even set panic on Not Maskable Interrupt
  ```
  echo 1 > /proc/sys/kernel/panic_on_unrecovered_nmi
  ```

Kernel and High-Availability
Channel Bonding driver

Channel Bonding

- The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface
- The behavior of the bonded interfaces depends on the bonding mode configuration
- Link integrity monitoring may be performed
- Channel bonding can be used to grant high availability

Configuration

- Enable in the kernel under Network device driver menu *Bonding driver support*
- In user space channel bonding can be configured in two ways:
  - ifenslave command
  - via sysfs

Bonding Modes (1)

- Channel bonding can be configured in several modes:
  - Balance-rr: round-robin over the slave interfaces
  - Balance-xor: transmit based on the selected transmit hash policy. Default is a simple xor between source MAC and destination MAC
  - Broadcast
  - 802.3ad: IEEE 802.3ad Dynamic link aggregation
  - Balance-tlb: outgoing traffic is distributed according to the current load on each slave. If the receiving slave fails, another slave takes over the MAC address of the failed receiving slave

Bonding Modes (2)

- Balance-alb: balance-tlb plus receive load balancing (rlb) for IPV4 traffic
- Active-backup: only one slave in the bond is active. A different slave becomes active if, and only if, the active slave fails

Bonding driver option (1)

- arp_interval
  Specifies the ARP link monitoring frequency in milliseconds
- arp_ip_target
  Specifies the IP addresses to use as ARP monitoring peers when arp_interval is > 0
- arp_validate
  Specifies whether or not ARP probes and replies should be validated in the active-backup mode
- downdelay
  Specifies the time, in milliseconds, to wait before disabling a slave after a link failure

Bonding driver option (2)

- fail_over_mac
  Specifies whether active-backup mode should set all slaves to the same MAC address at enslavement (default), or, when enabled, perform special handling of the bond's MAC address
- miimon
  Specifies the MII link monitoring frequency in milliseconds
- num_grat_arp
  Specifies the number of gratuitous ARPs to be issued after a failover event

Bonding driver option (3)

- primary
  A string (eth0, eth2, etc) specifying which slave is the primary device
- updelay
  Specifies the time, in milliseconds, to wait before enabling a slave after a link recovery
- use_carrier
  Specifies whether or not miimon should use MII or ETHTOOL ioctls vs. netif_carrier_ok() to determine the link status

Configuring for HA (1)

- Channel Bonding example configuration with ifenslave command and two interfaces:

```
modprobe bonding mode=active-backup \
arp_interval=100 \
arp_ip_target=192.168.2.8,192.168.2.10
modprobe e1000
ifconfig bond0 192.168.2.1 netmask 255.255.255.0 up
ifenslave bond0 eth0
ifenslave bond0 eth1

To remove:
ifconfig bond0 down
rmmod bonding
rmmod e1000
```

Configuring for HA (2)

- Channel Bonding example configuration with sysfs and two interfaces:

```
modprobe e1000
echo active-backup > /sys/class/net/bond0/bonding/mode
ifconfig bond0 192.168.2.1 netmask 255.255.255.0 up
echo +192.168.2.8 > /sys/class/net/bond0/bonding/arp_ip_target
echo +192.168.2.10 > /sys/class/net/bond0/bonding/arp_ip_target
echo 100 > /sys/class/net/bond0/bonding/arp_interval
echo +eth0 > /sys/class/net/bond0/bonding/slaves
echo +eth1 > /sys/class/net/bond0/bonding/slaves
```

Bonding proc file

- Each bonding device has got a file residing in the /proc/net/bonding directory. The file includes information about the bonding configuration, options and state
- For example:

```
Ethernet Channel Bonding Driver: v3.3.0 (June 10, 2008)
        Bonding Mode: load balancing (round-robin)
        Currently Active Slave: eth0
        MII Status: up
        MII Polling Interval (ms): 1000
        Up Delay (ms): 0
        Down Delay (ms): 0

        Slave Interface: eth1
        MII Status: up
        Link Failure Count: 1

        Slave Interface: eth0
        MII Status: up
        Link Failure Count: 1
```

Legal statements

- Linux is a registered trademark of Linus Torvalds
- Linux logo by Larry Ewing
- Other company, product, and service names may be trademarks or service marks of others