Challenging the world to change
heartbeat deployment tutorial

Lars Marowsky-Brée
SUSE Labs
Introduction

- Overview and concepts of High Availability
  - Introduction
  - Concepts
- heartbeat itself
  - Overview
  - Capabilities
  - drbd et al
  - Example configuration
- Other rants and topics
- Questions
Who is in the audience?

- **Jobs:**
  - System administrators
  - Developers
  - Users
  - Business people

- **Experience**
  - First timers
  - Experienced
  - Gurus
    - Go leave now ;-)
High Availability debuzzed

- Faults **will** occur – Murphy cannot be tricked.
- Unmasked faults show through to the users and cause downtime (and eventually loss of money)
- Some faults can be effectively masked completely ("fault tolerant", "continuous availability") and some only can be recovered from quickly ("high availability")
- Redundancy is important.
- HA is **not** perfect and **not** the same as Continuous Availability.
Some numbers

- Most people only take into account unscheduled downtime
  - ... scheduled downtime would also include the time needed for any updates or other system maintenance!

- Systems can „tolerate“ a certain number $k$ of independent failures ($k$-reliable) before total system failure.

- It's all about lies, damned lies and statistics

<table>
<thead>
<tr>
<th>Availability percentage</th>
<th>Yearly downtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>99,99999%</td>
<td>3s</td>
</tr>
<tr>
<td>99,999%</td>
<td>30s</td>
</tr>
<tr>
<td>99,999%</td>
<td>5m</td>
</tr>
<tr>
<td>99,99%</td>
<td>52m</td>
</tr>
<tr>
<td>99,9%</td>
<td>9h</td>
</tr>
</tbody>
</table>
The pyramid

- Life support
- Enterprise / Datacenters
- Telco / Carrier
- SMB / SoHo

Cost

Potential market size
Increase reliability

- Use a stable operating system
  - Write bugfree code!

- Reduce complexity - **KISS**!
  - Increase testing coverage
  - QA-aware processes for development and release

- Contain faults
  - "Failfast": suicide on uncorrectable faults turn **hard** byzantine faults into simple full failures
    - Deadman, hangcheck, panic on IO errors etc

- Find faults before they occur (**Orb of Detection anyone?!**)
  - Soft- and hardware monitoring
Reduce MTTR

- Introduce redundancy and eliminate SPoFs!
  - RAID redirects requests to healthy disks
  - Failover quickly reallocates service on healthy server
  - Load balancing redirects requests to healthy servers
  - Site failover eventually abandons site, backup data center goes live
- Needs to have redundant setups to have somewhere to move to.
  - States without redundancy are called „degraded modes“
  - Failed systems thus need to be repaired quickly still.
- Introduce redundancy and eliminate SPoFs!
- Also:
  - Service-level agreements, quick hardware replacement, good documentation
Single Points of Failure

- Infrastructure
  - Power
  - Air conditioning
  - Cabling in general
  - A single data center site

- Networking
  - Connection to the Internet
  - Intranet links
  - Firewalls

- Servers
  - Fans
  - Disks
  - Network cards

- Software
  - Applications
  - The cluster manager ;)

- Meatware
  - One administrator
  - No documentation
Clusters

- A cluster groups resources to form a system beyond the capabilities of a single node.

Availability
- Combined reliability of all nodes
- Services remain up and running even if a single node fails
- Remaining nodes take over

Performance
- Combined power of all nodes operating in parallel.
- Distribute load among cluster members
- Scalable systems
Cluster styles

- **Failover**
  - ... is like a good magicians trick:
  - **The hand moves faster than the eye**
  - Failover within a cupboard, rack or datacenter
  - Site-failover to protect against disasters
    - Different building or different continent...

- **Load-balanced server farms**
  - Provide load distribution, as well as protection against failed servers by shifting load away from them
  - Linux Virtual Server
Distributed Systems

- A number of nodes interconnected via some network
  - "Partially synchronous"

- Nodes:
  - Will fail, should try to contain faults locally
  - Assumed to have local stable storage
  - Randomly freeze, continue, do not run in step lock

- Certain assumptions about network reliability, performance
  - Links fail, packets get lost, the network gets partitioned
  - Packets get reordered, delayed, corrupted or messed with
Cluster Membership

- Cluster needs to have a consistent view of which nodes are valid members
- Nodes usually heartbeat actively or are otherwise monitored
- Lack of heartbeats received will lead to assumption of a node failure
- Difficult to distinguish between link failures, node failures if all links are cut – leads to *split-brain scenarios*
- Because failures can only be detected asynchronously, the cluster state is **never** certain.
Messaging in clusters

- Messages are transported via the network...
- $k$-reliable: At any phase of the protocol, $k$ failures are tolerated and recovered, thus masked for the higher levels
- Ordered: Global ordering is enforced.
  - Total order: All nodes see all messages in the same order.
  - Causal order: A message is only seen if all messages it might depend on are received.
- Cryptography protects against (malicious) corruption.
- Fun: Network partitioning and byzantine failures
Time in clusters

- With unsynchronized clocks, events could have totally random timestamps.
- Partially synchronized clocks (NTP) allow the assumption that any two clocks do not differ by more than $\Delta t$
- To simulate strictly monotonous time, systems would never be allowed to produce a timestamp higher than that of an event received from another node. ("causal" time)
- Foundation on this theory done by Leslie Lamport
Resource Management

- **Resource**: Encapsulation of physical/logical entity providing a given service
  - *Resource type*: IP address, mounted partition, httpd, Oracle...
  - *Resource Instance* identified by type, name and other instance parameters

- A **Resource Agent** script handles resources of a given type, providing a common API to the **Resource Manager**

- Typically only one node can run a given resource instance at any given time, allocating it on multiple nodes will lead to data corruption and **must be avoided**.

- Typically grouped into **resource groups**
  - Stack of resources needed to provide a given service
Crown jewels

- The data must be protected.
- Without data, there is no service.
- Without data, there was no service.
- Without data, there will be no service.

Data corruption must be prevented at all costs.
- Rather have no running service than potentially corrupt data.
  - Exceptions exist, but this is the safe default.
- Disconnect failed nodes from access to storage: Fencing
Accessing the data

- Shared storage via
  - Fibre Channel
  - Shared SCSI
  - iSCSI / HyperSCSI
  - Software replication

- Increasing hw reliability
  - Replication
  - Multi-path
  - RAID

- Increasing sw reliability
  - Software RAID
  - Replication
  - Journaled filesystems
  - Logical Volume Management

- Concurrent access
  - SAN-aware filesystems
    - GFS, PolyServe, IBM GPFS
  - Distributed filesystems
    - NFS, Lustre, AFS
IO fencing and STONITH

- Node-level fencing
  - Shot the other node/machine in the head (aka "STONITH")
  - Crude, but simply effective.
  - A powered down node does not modify data.

- Resource-level fencing
  - Resources get to fence themselves by excluding non-members
  - Disabling the Fibre-Channel switchport
  - Self-fencing RAID controllers
  - SCSI reservations
    - Caveat, emperor: Dangerous in multipath environments, not always reliable
Kinds of failover systems

- **Cold**
  - Hardware ready, but not running
  - Services are manually started on the new system
  - Downtime very noticeable

- **Warm**
  - System up and running, data kept in sync
  - Services are started automatically
  - Short outage is usually noticeable ("fast reboot")

- **Hot**
  - Up and running
  - All systems active
  - Client talks to multiple systems at once ("transaction monitor")
  - No noticeable outage

**Why isn't everything Hot then?**
- Expensive
- Software has to be adapted.
Failover summarized

- Servers constantly heartbeat each other.
- A node fails to send heartbeats for a given period of time.
- Other node(s) detect this, assume node has failed.
- Node is prevented from corrupting the data by being fenced off.
- Membership computed anew.
- New node is chosen, services are reallocated there.
- Called „switchover“ if not triggered by a failure, for example for taking a node out of the cluster for maintenance.
- „Failback“ is either done automatically or manually initiated.
heartbeat overview I

- Started by Alan Robertson in 1999
  - Stable branch 1.2.0 (*1.0.4 still supported*)
  - Development branch 1.3.0
- 2 node warm failover for „resource groups“
  - Either automatic failback or controlled.
- Simple to setup
- Security focus, compact code base
- Redundant links for heartbeating and communication
  - Serial + IPv4 based unicast, broadcast, multicast
  - „ping“ nodes can be added as pseudo-cluster members
heartbeat overview II

- Locked into memory and soft realtime for minimal latency
- Sub-second node death detection possible with version ≥1.2

- Contains many interesting components:
  - Ipfail can monitor external connectivity
  - Consensus Cluster Membership layer for up to $N$-nodes
  - STONITH modules, IPC library, PILS library
  - Simple, mostly reliable cluster communication
  - Cluster Test System (CTS) included
  - Application heartbeating, data checkpointing etc
Heartbeat limitations

- Resource management limited to two nodes
- Resources themselves not monitored for failure
- Configuration not automatically synchronized
- Little real support for replicated resource types
heartbeat future

- Multi-node functionality
- Dependency based resource model with constraint solver.
  - Internal model, users will probably need a helpful GUI
- Automatically replicated configuration (Cluster Information Base)
- $k$-reliable, causal/total ordered messaging service.
- Resource Agents conforming to OCF RA API
  - Resource monitoring will be extended
- Ready when it is ready, but work is underway.
Resource Agents

- Allow heartbeat to manage a given resource type
- Heartbeat also supports using init.d scripts as Resource Agents
- Typically written as shell scripts
- Take simple list of options plus a requested action

Key points to keep in mind:
- Resource agents need to be able to recover from an unclean shutdown
- Be extra careful.
- For more details, refer to Open Clustering Framework Resource Agent API
Currently available RAs

- IPaddr, IPaddr2
- IPsrcaddr
- Filesystem
- LVM
- RAID1
- ServeRAID
- ICP
- LinuxSCSI
- portblock
- xinetd
- db2
- WAS
- Mostly all init.d scripts
- It is easy to add your own.
STONITH modules

- Network power switches:
  - APC MasterSwitch
  - WTI NPS/TPS
  - BayTech RPC-xxx
  - Night/Ware RPC100S

- Serial power switches:
  - WTI RPS
  - APC Smart UPS

- IPMI over ethernet
- For testing only:
  - ssh
  - meatware
    (Manual operation)
ipfail

- heartbeat can monitor external nodes as pseudo cluster members by pinging them
- ipfail uses that information to determine where to place resources, namely on the node with the better external connectivity
- Dead simple configuration:
  - Configure ping nodes
  - Tell heartbeat to start ipfail
Cluster Test System

- CTS is the heartbeat test harness, stable releases are required to pass a few thousand iterations before release.
- Can be extended with your own tests! (send patches!)
- Requires a test-controller which orchestrates the tests on the two victim nodes.
- syslog may lose packets.

- BasicSanityCheck as a single node, straightforward sanity check of the compiled system.
- No hardware? No problem:
  - User-Mode-Linux
  - VMware
Top 7 rules

1. Keep it simple, silly!
2. Redundancy.
3. Read documentation.
4. Write documentation.
6. Test.
7. Redundancy.
Common mistakes

- Not having the configuration files synchronized between all nodes.
- Controlling resources in the system init scripts and not under heartbeat's control.
- Not using STONITH when required.
- Nodenames are case-sensitive.
- Cheap serial cables.

- Non-redundant heartbeat media.
- Resource Agents which do not report to "status" correctly.
drbd present

- Distributed Replicated Block Device by Philipp Reisner
- Storage replication over the network (like RAID1) between two nodes over an „arbitrary“ distance
- „Smart resync“ for all failure scenarios now, reducing MTTR
- Three protocols with different data integrity/performance trade-offs
- Benchmarks show that with GigE, performance is limited by spindle and wire speed, drbd overhead negligible
drbd illustrated
ha.cf example

keepalive 200ms
warntime 500ms
deaddtime 1s
initdeadt ime 30
auto_failback off
baud 19200
serial /dev/ttyS0
mcast eth0 225.0.0.1 694 1 0
bcast eth1
stonith_host node1 rps10 /dev/ttyS1 node2 0
stonith_host node2 rps10 /dev/ttyS1 node1 0
node node1 node2
ping 10.10.10.254
respawn hacluster /usr/lib/heartbeat/ipfail
haresources + authkeys

- haresources
  
  node1 10.10.10.11 datadisk::drbd0 \\
  Filesystem::/dev/nb0::/ha/apache::reiserfs apache

  node2 10.10.10.12 datadisk::drbd1 \\
  Filesystem::/dev/nb1::/ha/mysql::ext3 mysql

- authkeys must be root:root, mode 0600!

  auth 1

  1 shal MySeCrritPassword
drbd.conf

```
resource drbd0 {
  protocol = C
  fsckcmd = /bin/true
  inittimeout=0
  disk {
    do-panic
    disk-size = 4194304k
  }
  net {
    sync-min = 40M
    sync-max = 90M
    tl-size = 5000
    timeout = 60
    connect-int = 10
    ping-int = 1
  }
}
```

```
on node1 {
  device = /dev/nb0
  disk = /dev/hdc1
  address = 192.168.1.1
  port = 7788
}
on node2 {
  device = /dev/nb0
  disk = /dev/hdc1
  address = 192.168.1.2
  port = 7788
}
```

```
resource drbd1 {
  ...
}
```
Standards ... and so many!

- Interoperability does not exist.
- Open Clustering Framework as a community-driven approach
  - Very slow going.
  - Resource Agent API done, though
  - Working group of the FSG.
- Service Availability Forum as an „Industry driven“ initiative
  - Telco focus
  - Closed group, expensive membership
- CGL / DCL OSDL wishlists keep listing clustering
Other projects

- Other failover software:
  - RH ClusterManager
  - WitchDoctor WDX
  - IBM Tivoli System Automation for Linux
  - PolyServe

- Related topics:
  - Stateful firewall failover
  - Security
    - Denial of Service, data corruption, etc affect reliability
    - A cluster is basically a single administrative entity
    - „RAS“ of all sorts
    - Hotplug of memory, CPUs, you name it
Single System Image

- Implementation by HP / Bruce Walker, ported from Tru64 and older.
- All nodes form a virtual single system, moving the cluster boundary even below the administrator.
- Cluster-wide devices, cluster-wide memory (NC-NUMA) etc
- Mostly complete cluster infrastructure
- Fairly invasive, but still interesting.
Cluster Volume Manager

- **LVM2:**
  - Proprietary Cluster-extensions by Sistina

- **EVMS2:**
  - Cluster-extensions based on top of heartbeat, all Open Source
    - Actually EVMS project contributed the CCM to heartbeat for this usageage

- **Significantly** higher priority than CFS
  - Few apps can take real advantage of concurrent access to the same filesystem still, but CVM is a real pre-requisite to sensibly managing storage in a cluster
Linux Virtual Server

- Started by Wensong Zhang, very stable
- Load balances working on TCP/UDP layer
- Dedicated node acts as load balancer / firewall
  - Need two of them, too.
- Virtual IP mapped via a scheduler to a set of real servers and then forwarded via NAT, „Direct Routing“ or tunneling
- Various scheduling methods
- Can replicate state table from a master LB to a secondary
- Layer 7 load balancing (*ktcpvs*) in alpha quality
Cluster Alias

- Load balancing without a load balancer
- All nodes bring up the same IP address – how can this work?
- Nodes all reply to ARP requests with a multicast MAC
- Only one node replies to an incoming new connection; other packets are silently ignored.
  - (src ip, port; dst ip, port) can be used as a hash key, Cluster Manager reallocates hash buckets in case of failures
- IP address can be used for outgoing connections too
- Implementation being done right now
Software RAID issues

- Bootloader integration for RAID1 / multipath
- DM/MD/LVM2/EVMS2/hotplug integration still not fully designed
- 2.6 code base is mostly a rewrite, experience from 2.4 suggests substantial testing and debugging efforts
md codebase in 2.4 sucks
MD codebase in 2.6 doesn't suck as badly
Used for local replication & redundancy
Up to one fault per RAID group can be masked for RAID>0
MTTR can be reduced by added hotspares
Multipathing can protect against certain media failures
RAID1 on top of multipathing can provide „fault tolerance“
RAID6 can mask 2 failures per array and increases space-efficiency for bigger arrays
drbd vs nbd/md

- drbd is more tightly integrated with the version vector; failures like split-brain, one node rebooting etc do not occur in md scenarios
- Close integration with the network layer makes the protocol A and B possible, which would be rather difficult to add to nbd
- md+nbd is more complex to setup
- Possibility would be to add appropriate scripting around md+nbd to ease this, but nobody steps forward to do that ;-(}
Further reading

- *In search of clusters* by Gregory F. Pfister
- *Blueprints in High Availability* by Evan Marcus, Hal Stern
- *Fault tolerance in Distributed Systems* by Pankaj Jalote
- *Distributed Algorithms* by Nancy Lynch
- *Transaction Processing* by Jim Gray
- [http://www.lcic.org/](http://www.lcic.org/) for more links to projects
- [http://linuxha.trick.ca/](http://linuxha.trick.ca/) Linux HA Wiki
- [http://linux-ha.org/](http://linux-ha.org/)
No questions!^H?